Pumping Tests In Fractal Media

Heiko Pälike

Thesis submitted as partial fulfilment of the requirements for the
M.Sc. in Hydrogeology and Groundwater Resources
University College London

September 1998

Abstract

The investigation of fractured rocks and their hydraulic properties has recently received growing attention as the result of plans to build nuclear waste repositories. Studies have shown the need to refine existing methods of measuring hydraulic parameters of fractured rocks, especially for contaminant transport problems. Traditional approaches have generally tried to model hydraulic tests in fractured media, first by conventional type curve analysis and then using more and more sophisticated models if the match of analytical predictions was poor with field data. In some instances it is possible to obtain a good match by, e.g., using a leaky aquifer model, even though the parameters obtained might not reflect the underlying geometric properties of the aquifer, leading to misleading interpretations.

This study focused on the combination of two novel approaches of hydraulic pumping tests in fractured media, using recently developed theories of flow in *fractal media*, and *inverse modelling:*

1. A theoretical study investigating the properties of regular and stochastic fractal fracture networks. The main aim is to study functional relationships between fractal dimension measures of these networks, fractional flow dimensions, network connectivity and fracture probability in order to condition stochastic fracture networks. This is a first step to model the pumping response making use of the underlying fractal structure of fractured rocks. Computer codes and algorithms were developed and adapted to generate two-dimensional fracture networks with fractal properties using an iterated function system. Code was developed to prepare these networks for input into a finite element code so that the fractional flow dimension can be obtained. Results indicate that connectivity is related to flow dimension, and also to the network topology ("diffusion slowdown"). Calibration curves were generated that relate fractal dimension, flow dimension, fracture probability and connectivity.
2. Another part of this study was to test the usefulness of a direct inverse modelling approach ("simulated annealing"). This method tries to fit existing pumping test data by altering the conductivity properties of a network input "template" (e.g. a generic 2-D regular mesh), without the intermediate step of parameter estimation using traditional type curves. A new approach was the combination of this method with the fractal fracture network generator mentioned above. Computer programmes were developed to inverse model synthetic pumping test data generated from fractal networks. It was revealed that inverse modelling runs generate networks with similar fractal properties as the one that was used to generate the pumping test data. The actual appearance of networks, however, can be radically different. Most importantly, by generating several instances of networks for the same input data it is possible to obtain a *measure of uncertainty*, with networks showing similar connectedness in regions of low uncertainty and large variations in other regions, enabling to determine how well different wells are connected, and where more data need to be collected to allow meaningful predictions

In addition to the results obtained, the developed computer code will be of potential great use for the research group, enabling further work to study fractal media. The simulated annealing method and fractal network algorithms can be easily extended to 3-D and should be used in future work as soon as real data sets and enough computing power become available. A possible application of this method would be the interpretation of the NIREX RCF3 long

term pumping test data set (requested data from NIREX did not arrive in time for this project).

Acknowledgements

Professor John Barker, my project supervisor, is greatly acknowledged for providing an incredible wealth of interesting ideas, literature, patience, and very important input.

The Natural Environment Research Council (NERC) for sponsorship of this course.

Julie Najita (Lawrence Berkeley Laboratory) for providing the inverse modelling code and Jorge Acuna (Unocal) for providing code and help for iterated function systems.

Mike Streatly (Entec) for a very good presentation on the pumping tests in the Sherwood Sandstone Group as part of the NIREX RCF3 Nirex long term pumping test. Maybe the data will be used next year.

Golder Associates for information on the NIREX RCF3 long term pumping test.

Table of Contents

1. INTRODUCTION 1

2. FRACTALS, FLOW DIMENSION AND FRACTAL AQUIFER PROPERTIES 4

2.1 Introduction 4

2.2 Non-integer flow dimension 4
2.2.1 Background 4
2.2.2 Mathematical description 7
2.2.3 Pumping well pressure response 8
2.2.4 Observation well pressure response 9

2.3 Fractal theory in geology 11
2.3.1 Definition of fractals 12
2.3.2 Fractal geometry 13
2.3.3 Self-similar, self-affine and statistically self-similar fractals 18
2.3.4 How to measure fractal dimension 20
2.3.5 Fractal dimensions for different Euclidean embedding dimensions 25
2.3.6 The problem of multifractals and lower/upper cut-off length scales 27

2.4 Review of fractal structures in hydrogeology 29
2.4.1 Introduction 29
2.4.2 Pore geometry 31
2.4.3 Fracture surfaces 33
2.4.4 Mapping of fault traces and lineaments 34
2.4.5 Fractal relationship between fault number, lengths and widths 35
2.4.6 Fractal distribution of three-dimensional fracture networks 36
2.4.7 Other applications of fractal analysis 38

2.5 Pressure-transient analysis of fractal reservoirs 39
2.5.1 Introduction 39
2.5.2 Theory of flow equations in fractal media 39
2.5.3 Discussion of the parameter □ 49

3. FORWARD MODELLING OF FRACTAL AQUIFER RESPONSE 52

3.1 Generation of synthetic fractal fracture networks 52
3.1.1 The iterated function system (IFS) approach 52
3.1.2 A modified Sierpinski gasket 54
3.1.3 Flow area properties of recursive networks 55

3.2 Simulating a pumping test on a regular fractal network 56

3.3 A stochastic fractal pattern generator 65
3.3.1 The relationship between fracturing probability and fractal and flow dimension 67

4. INVERSION MODELLING USING SIMULATED ANNEALING 73

4.1 Introduction to simulated annealing 73

4.2 Origin of the simulated annealing algorithm 74

4.3 Description of the simulated annealing algorithm 75

4.3.1 The hiker analogy 75
4.3.2 The Metropolis criterion 75

4.4 Adaptation for flow in fractured media 78
4.4.1 A description of the possible system configuration 79
4.4.2 The perturbation step 81
4.4.3 The cooling schedule 81

4.5 Technical implementation 82
4.5.1 Fracture network model 82
4.5.2 Energy calculation 83
4.5.3 Cluster variable aperture perturbation 83

4.6 Simulated annealing evaluated on simple networks 84

4.7 Inverse modelling from fractal pressure transient data 87

5. CONCLUSIONS 98

6. RECOMMENDATIONS 101

7. CITED REFERENCES 103

8. EXTENDED BIBLIOGRAPHY 110

List of Appendices

Appendix A - Excel functions to compute fractal pressure-transient data
Appendix B - Code in C to generate modified Sierpinski gasket networks
Appendix C - Post-processing of fracture network files

List of Figures

Figure 1: The relationship between flow dimension and scaling of cross-sectional flow area. 6
Figure 2: Type curves for the generalised radial flow model at the pumping well (r_D=1). 10
Figure 3: Type curves for the generalised radial flow model (r_D=100). 10
Figure 4: A one-dimensional line. 13
Figure 5: A two-dimensional plane. 13
Figure 6: A three-dimensional space. 14
Figure 7: The construction of the Koch "snowflake" or curve. 16
Figure 8: The triadic Koch curve and the "Devil's staircase" illustrating self-affine and self-similar sets. 18
Figure 9: Brownian motion of several particles in 2-D. 19
Figure 10: Illustration of how fractal dimension is defined by the slope of a line on a log-log plot. 20
Figure 11: Two- and three-dimensional modified Sierpinski gasket, Sierpinski pyramid and the Menger Sponge 26
Figure 12: Measured relationship between core plug air permeability, apparent similarity dimension D_S and area shape factor S_a 31
Figure 13: Examples of fractal fracture surfaces 34
Figure 14: Mass scaling on different networks with 2-D embedding dimension. 40
Figure 15: The effect of the parameter □ on the pressure-transient response. 46
Figure 16: Illustration of the iterative generation of a self-similar network. 54
Figure 17: Modified Sierpinski gasket with 3 fracture generations. 55
Figure 18: The modified Sierpinski gasket used to simulate flow in a fractal network. 59
Figure 19: Log-log plot of simulated drawdown vs. time at the pumping well. 60
Figure 20: Log-log plot of drawdown and drawdown derivative w.r.t. log time versus time for the observation well closest to the pumping well. 61
Figure 21: Log-log plot for a second observation well. 62
Figure 22: Log-log plot for a third observation well. 63
Figure 23: Log-log plot for a fourth observation well. 64
Figure 24: Illustration of four fractal networks with different fractal mass dimensions, generated by varying fracturing probability. 66
Figure 25: Three plots corresponding to the data in table 3, defining a network fractal dimension by approximation of a straight line. 70
Figure 26: Similar to figure 25, this plot shows the correlation between various fractal dimension and fracturing probability. 71
Figure 27: This plot shows the relationship between the fractal " information" dimension of generated networks and the slope of the pumping well pressure transient. 72
Figure 28: Configurations of a 100-city travelling salesman problem obtained after 0, 68, 92 and 123 steps of the algorithm. 73
Figure 29: The input grid used to generate synthetic drawdown transients. 85
Figure 30: The transmissivity distribution obtained by inversion of the synthetic pumping test data. 86

Figure 31: This plot shows the gradual improvement in fit between model and input drawdown data for the model shown in figure 30. 86
Figure 32: The fractal network that was used to generate synthetic pumping test data. 88
Figure 33: A log-log plot of drawdown versus time showing the pressure transient data. obtained from the pumping well and the seven observation wells. 90
Figure 34: The 'template' network used for inversion. 91
Figure 35: Six realisations of networks obtained by inversion. 92
Figure 36: This log-log plot shows the drawdown and drawdown derivative w.r.t. time at the pumping well of the original grid and the six inversion runs. 93
Figure 37: An "average" representation of the six output grids obtained by inversion. 94
Figure 38: The radial variation of the fractal measure "fracture intersections / cumulative fracture volume". 97

List of Tables

Table 1: A selection of methods to obtain a fractal dimension for self-similar and self-affine sets. 21
Table 2: Measured fractal dimension for different geological applications. 30
Table 3: Values obtained for capacity dimension, information dimension and correlation dimension for different values of fracturing probability. 69
Table 4: Values of fractal dimension, slope of pumping response, network size and fracturing probability for a network with 14 fracture generations. 71
Table 5: Summary of parameters used to generate the synthetic pressure transient data. 88

1. Introduction

It has become clear in recent years that more theoretical work is necessary to adequately describe and characterise flow in fractured media, particularly to increase the accuracy of existing methods in order to enable accurate predictions of spatial as well as time properties of groundwater flow. The investigation of fractured rocks and their hydraulic properties has recently received growing attention as the result of plans to build nuclear waste repositories. Studies have shown the need to refine existing methods to measure hydraulic parameters of fractured rocks, especially for contaminant transport problems. Traditional approaches have generally tried to model hydraulic tests in fractured media first by conventional type curve analysis and then using more and more sophisticated models if the match of analytical predictions was poor with field data. In some instances it is possible to obtain a good match by, e.g., using a leaky aquifer model, even though the parameters obtained might not reflect the actual geometric properties of the aquifer, leading to misleading interpretations. In other cases it has also been observed that none of the existing models can fit the data (Black et al., 1986).

As part of the course work for the M.Sc. in Hydrogeology and Groundwater Resources at UCL the use of pumping tests to characterise aquifers was dealt with exhaustively. The author realised that traditional methods of type-curve matching always require input from the hydrogeologist, often in the form of an “educated guess”, to choose an appropriate conceptual model, flow geometry, flow dimension, and effectively making assumptions on the nature of the aquifer that should be determined from a test. Of course, this is the result of the possible non-uniqueness of inverse modelling from pumping test data (Sun, 1994) and this also explains the value of an experienced hydrogeologist. However, the author felt that there should be more scientific and sophisticated methods available to parameterise an aquifer. Ideally, a method should honour and incorporate all existing data (e.g. pumping tests, fault mapping, geophysics, tracer tests etc.) and give a measure of uncertainty indicating where more data need to be collected.

Preliminary experiments where undertaken to see whether slightly more sophisticated methods such as the automatic fitting of type curves by minimising the misfit between observation data and model data are feasible. However, it was quickly found that even by

using sophisticated minimisation routines such as the Levenberg-Marquardt method, automated fitting procedures only find a local minimum in most cases and still require many assumptions.

The author originally intended to analyse data sets that were obtained during the RCF3 long term pumping test in Sellafield by Nirex, the company set up by the government to deal with the disposal of nuclear waste in Britain. These data promised to show interesting properties, which would enable the verification of newly developed theories for flow in fractured, sparsely connected rocks. Unfortunately the data needed did not arrive in time for this project. Instead, the author concentrated on more theoretical aspects of flow in fractured media.

This study focused on the combination of two novel approaches of hydraulic pumping tests in fractured media, using recently developed theories of flow in *fractal media*, and *inverse modelling:*

1. A theoretical study investigating the properties of regular and stochastic fractal fracture networks. The main aim is to study functional relationships between fractal dimensional measures of these networks, fractional flow dimensions, network connectivity and fracture probability in order to condition stochastic fracture networks. This is a first step to model the pumping response making use of the underlying fractal structure of fractured rocks. Computer codes and algorithms were developed and adapted to generate two dimensional fracture networks with fractal properties using an iterated function system. Code was developed to prepare these networks for input into a finite element code so that the fractional flow dimension can be obtained. Results indicate that connectivity is related to flow dimension and also to the network topology ("diffusion slowdown"). Calibration curves were generated that relate fractal dimension, flow dimension, fracture probability and connectivity.
2. As pointed out before, it was tried to find an analysis method that only requires minimal subjective input from the modeller. A part of this study was to test the usefulness of a direct inverse modelling approach using "simulated annealing". This method tries to fit existing pumping test data by altering the conductivity properties of a generic network

input "template", *without the intermediate step of parameter estimation* using traditional type curves. This method can also be adapted for other minimisation problems, including traditional porous media pumping tests. The great advantage of this method is that it honours all existing data, including deviations from theoretical models. Here, spatially distributed field data are directly used to obtain distributed parameter field of the aquifer. Code for inverse modelling was kindly supplied by Julie Najita, Lawrence Berkely Laboratory. A new approach was the combination of this method with the fractal fracture generator described above. Computer programmes were developed to inverse model synthetic pumping test data from generated fractal networks. It was revealed that inverse modelling runs generate networks with similar fractal properties as the one that was used to generate the pumping test data but that the actual appearance of networks can be radically different. Most importantly, by generating several instances of networks for the same input data it is possible to obtain a *measure of uncertainty*, with networks showing similar connectedness in regions of low uncertainty and large variations in other regions, enabling to determine how well different wells are connected and where more data need to be collected to allow meaningful predictions

In addition to the results obtained, the developed computer code will be of potential great use for the research group, enabling further work in the study of fractal media. The simulated annealing method and fractal network algorithms can be easily extended to 3-D and should be used in future work as soon as real data sets and enough computing power become available. A possible application of this method would be the interpretation of the NIREX RCF3 long term pumping test data set, data for which were not obtained in time for application in this study.

Note that a considerable amount of time available during the five months duration of this project was spent on developing and debugging computer software and that the remaining time limited the scope of further study. More work is clearly essential and can now be done using the software tools developed.

2. Fractals, flow dimension and fractal aquifer properties

2.1 Introduction

Recent experience has shown that some pumping tests in fractured rock cannot be analysed with traditional pumping test methods. In particular, it has become clear that the assumption of flow geometry inherent in type curve approaches, like the well know "Theis" curve for 2-D flow, is not justified for flow in fractured media (Black et al., 1986). It was shown that the flow system could not be satisfactorily described using one-, two- or three-dimensional models, both with and without using the traditional Warren-Root dual porosity concept.
The first section of this chapter presents how the concept of flow dimension can be extended from integers (one for flow in a pipe, two for radial flow and three for spherical flow) to non-integer values as developed by Barker (1988). Then it is shown how this idea can be linked to the more general field of "fractal" theory and a review is given of advances of fractal applications in geology.

2.2 Non-integer flow dimension

2.2.1 Background

So far, flow geometry has not attracted special attention because sedimentary aquifers are normally approximated by a two-dimensional porous medium system where the aquifer height is negligible compared to the lateral extent and flow is largely radial. Within the aquifer, vertical flow variation is usually assumed to be negligible and the scale of heterogeneity is assumed to be larger than the scale involved in field testing (Black, 1994). This approach might be valid for homogeneous sedimentary aquifers, but it is not appropriate for fractured aquifers, where the main flow does not take place in the small matrix pore space but within fracture-planes and fracture channels.
It is intuitive that if a source zone in a borehole intercepts a single, constant diameter open tube, then a linear (one-dimensional) flow response will occur (i.e. a straight line on a linear-linear plot of drawdown versus time). The cross-sectional area through which flow occurs remains constant as the pumping progresses (see figure 1). A borehole that intersects a sparse,

poorly connected fracture system consisting mainly of constant aperture planar fractures might result in a system better described by one or two dimensional flow, whereas, if the fracture density is large and the distribution is isotropic, a three-dimensional spherical flow model would be more appropriate (Barker, 1988). Depending on the complexity of the system, a proper description of flow might involve an intermediate flow dimension, for example an equivalent flow dimension between one and two. The important feature in all cases is the *functional dependency of flow area versus radial distance* which, after inspection of figure 2, can be generally expressed as

$$A \quad r^{(D_f - 1)} \qquad \text{.....(1)}$$

where

A = cross-sectional flow area

r = radial distance from the source

D_f = flow dimension.

It becomes clear from the large variety of possible flow conditions that traditional flow models implicitly assume a certain flow dimension (e.g. two in the case of Theis) that is not appropriate and may lead to significant interpretation error. Barker (1988) developed a mathematical description of "generalised radial flow" that does not implicitly assume a flow dimension a priori. Instead, flow dimension is another system parameter that has to be determined from the available data.

The generalised radial flow model assumes that transient radial flow occurs in a medium that fills an n-dimensional space. Darcian type flow occurs in the homogeneous and isotropic flow medium described by hydraulic conductivity K and specific storage S_S. The source is an n-dimensional sphere, i.e. a cylinder for n=2 and a sphere for n=3. The generalised radial flow model can be applied to fractured and non-fractured media that show fractional space-filling characteristics (Doe, 1991). This is an important concept and will be discussed further in the section on fractal theory.

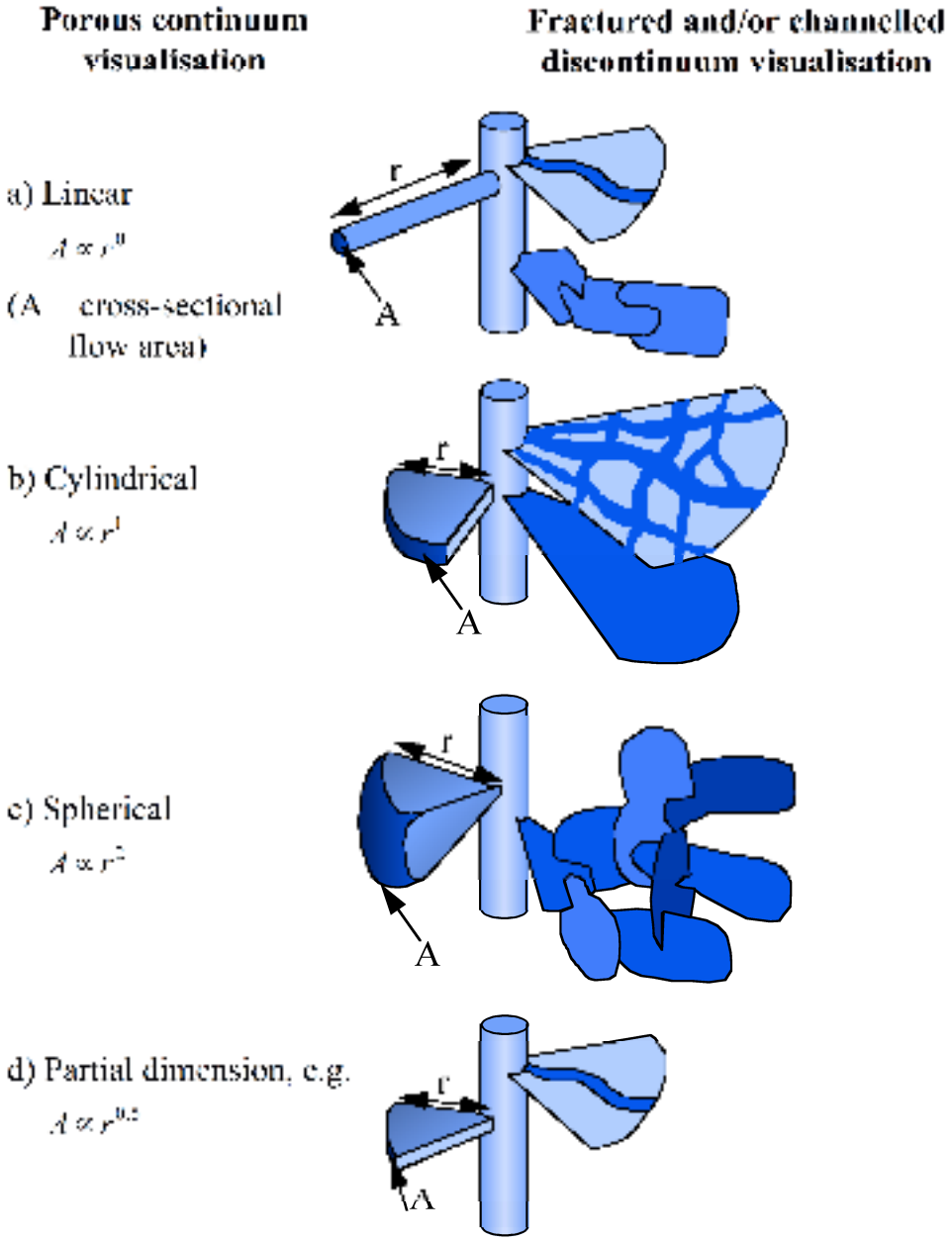


Figure 3: The relationship between flow dimension and scaling of the cross-sectional flow area (after Black, 1994). Note that d) as drawn by Black is not strictly correct since the area as shown still scales as r^1. However, the case with A $r^{0.5}$ is hard to visualise. See also figure 19.

2.2.2 Mathematical description

This section briefly reviews the theory behind the generalised radial flow model. For practical purposes, only equations 15, 16, 19 and 20 are important, from which appropriate type curves can be generated and then used to estimate flow dimension and other parameters (see Appendix A for the technical implementation with Excel macros).

For a constant-rate hydraulic test, Barker's (1988) fluid flow, written in dimensionless variables h_D (dimensionless head), r_D (dimensionless radius) and t_D (dimensionless time), is:

$$\frac{\partial^2 h_D}{\partial r_D^2} + \frac{n-1}{r_D}\frac{\partial h_D}{\partial r_D} = \frac{\partial h_D}{\partial t_D} \quad \text{..... (2)}$$

where

$$h_D(r_D, t_D) = \frac{Kb^{3-n}\alpha_n r_w^{n-2}}{Q} h(r,t) \quad \text{..... (3)}$$

$$t_D = \frac{Kt}{S_S r_w^2} \quad \text{..... (4)}$$

and

$$r_D = \frac{r}{r_w} \quad \text{..... (5)}$$

where b is the extent of the flow region, r the radial distance from the pumping well, r_w the well radius and α_n, the area of a unit sphere in n dimensions, is given by

$$\alpha_n = \frac{2\pi^{\frac{n}{2}}}{\Gamma(n/2)} \quad \text{..... (6)}$$

Γ denotes the gamma function.

For an infinitesimal source and infinite flow region, using the following boundary conditions:

$$h_D(r_D,0) = 0 \quad \text{..... (7)}$$

$$h_D(r_D \to \infty, t_D) = 0 \quad \text{..... (8)}$$

$$r_D^{n-1} \left.\frac{\partial h_D}{\partial r_D}\right|_{r_D \to 0} = -1 \quad \text{..... (9)}$$

the solution is[1] (Barker, 1988):

$$h_D(r_D,t_D) = \frac{r_D^{2\gamma}}{2^{\gamma}(1-\gamma)}\Gamma(1-\gamma,u) \quad \text{for } \gamma<1 \quad \text{..... (10)}$$

where

$$u = \frac{r_D^2}{4t_D} \quad \text{..... (11)}$$

and

$$\gamma = 1-\frac{n}{2} \quad \text{..... (12)}$$

2.2.3 Pumping well pressure response

Since the incomplete gamma function of equation **(13)** can be expressed as (Abramowitz and Stegun, 1964):

$$\Gamma(1-\gamma,u) = \Gamma(1-\gamma) - \sum_{m=0}^{\infty} \frac{(-1)^m u^{m+1-\gamma}}{m!(m+1-\gamma)} \quad \text{..... (14)}$$

and the pumping well hydraulic head becomes linear on a log-log plot for large times and $\gamma > 0$, i.e. n<2, (see figure 4), an asymptotic linear approximation can be obtained from (10) and (14), given by Barker (1988):

$$h_D(r_D=1,t_D) = \frac{(4t_D)^{\gamma}}{2^{\gamma}\Gamma(1-\gamma)} \quad \text{for } \gamma>0 \quad \text{..... (15)}$$

---

[1] Note that this is equivalent to $h_D(r_D,t_D) = \frac{r_D^{2\gamma}}{2^{\gamma}(1-\gamma)}\int_u^{\infty}\frac{\exp(-u)}{u^{1+\gamma}}du$ for $\gamma<1$, since $\Gamma(a,x) = \int_x^{\infty}\frac{\exp(-u)}{u^{1-a}}du.$

Hence, a large time plot of the pumping well response against time will have a slope of α (which gives n) and an intercept (at ln(t)=0) of

$$A = \frac{Q}{2\pi K b^{3-n} \Gamma_n (1-\alpha)} \left(\frac{4K}{S_S}\right)^{\alpha} \quad \text{..... (16)}$$

However, as can be seen from figure 5, at large r_D the linear portion of the log-log plot of head at an observation well may not appear until late times. This might not be measured in real pumping tests, that often stop before this behaviour an be observed.

2.2.4 Observation well pressure response

Chakrabarty (1994) showed that, by using the head *derivative* group obtained from (10), the linear behaviour on a log-log plot of head versus time at $r_D > 0$ can also be approximated. From (10), the time-derivative of h_D, for fixed r_D is given by

$$h_D' = \frac{4^u \exp\left(-r_D^2/4t_D\right)}{2\pi(1-u)t_D^{1-u}} \quad \text{..... (17)}$$

Hence,

$$\ln\left(t_D^{1-\alpha} h_D'\right) = \ln\left(\frac{4^{\alpha}}{2\pi(1-\alpha)}\right) - \frac{r_D^2}{4t_D} \quad \text{..... (18)}$$

By plotting the observation well response as $t^{1-\alpha} h'$ vs. 1/t on a semi-log graph, further analysis is possible. The linear plot should then be described by a slope of

$$B = \frac{S_S r^2}{4K} \quad \text{..... (19)}$$

and an intercept at 1/t =0 of

$$C = \frac{4^{\alpha} Q}{2 S_S b^{3-n} \Gamma_n (1-\alpha)} \left(\frac{S_S}{K}\right)^{1-\alpha} \quad \text{..... (20)}$$

Hence, calculating α from the slope of the log-log plot of the pumping well response and the diffusivity K/S_S from (19), Kb^{3-n} can be obtained from equation 15 and $S_s b^{3-n}$ from equation 20. Chakrabarty (1994) also noted that the intercepts A and C are related by $C = A^{\alpha}$.

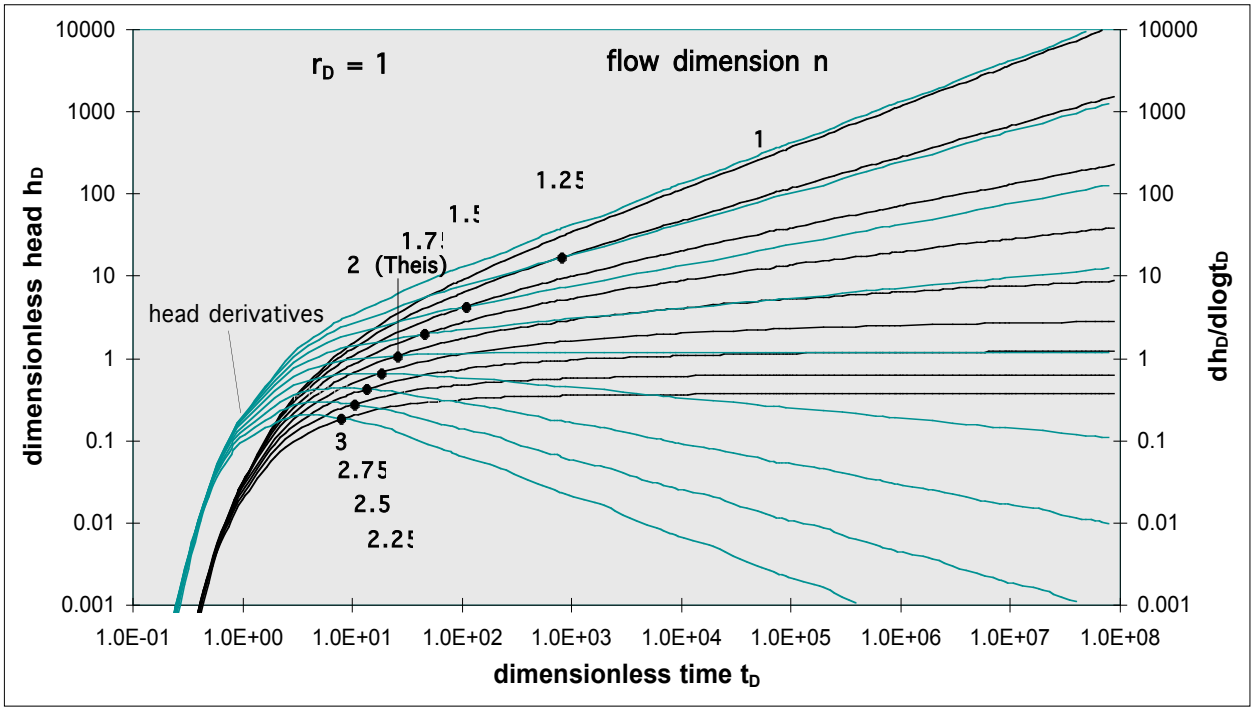


Figure 6: Type curves for the generalised radial flow model at the pumping well (r_D=1) and their derivatives w.r.t. log t_D. The curves were plotted with Excel macros (see Appendix A), according to equation 10. Instabilities in the numerical derivative method used explain the wiggles of the derivative curves, which should be straight lines at large times.

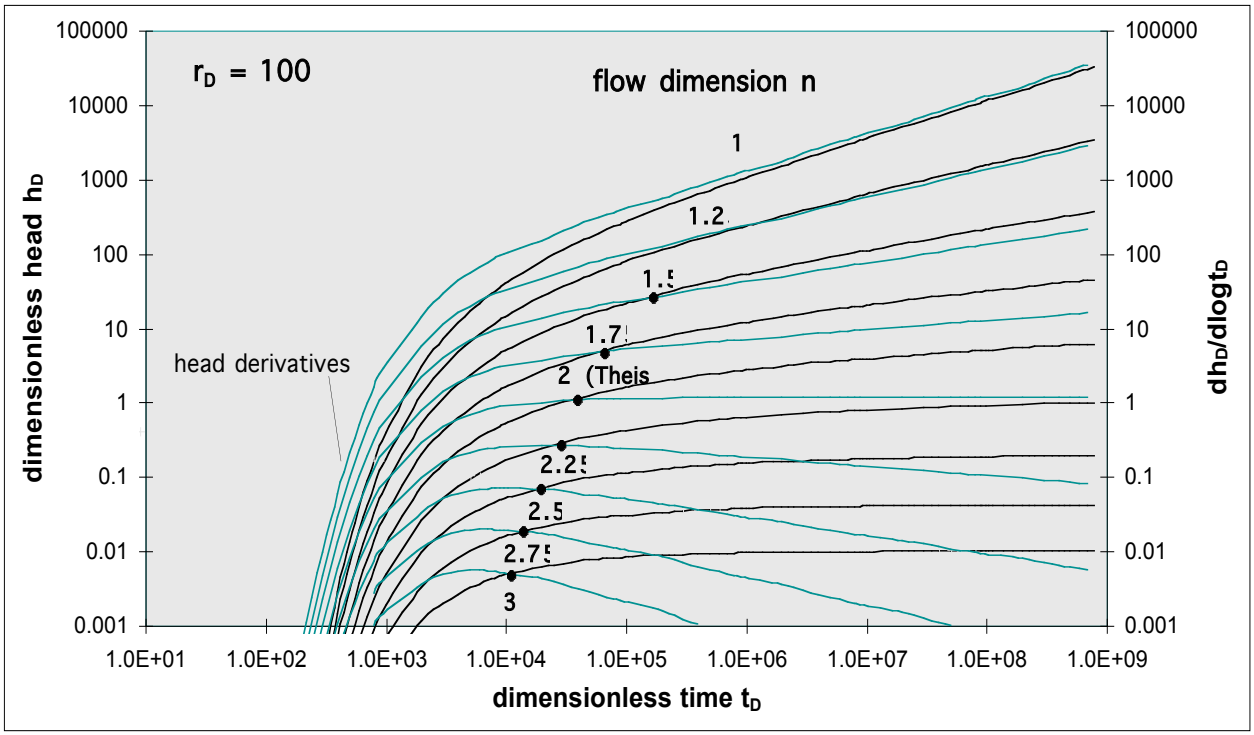


Figure 7: Type curves for the generalised radial flow model at r_D=100 and their derivatives w.r.t. log t_D. Note how the straight line behaviour only occurs at late dimensionless times.

2.3 Fractal theory in geology

It might not at first be clear why the previous section is linked to a discussion of fractal theory. It is important to note that Barker (1988) derived the generalised radial flow model without any reference to an underlying fractal structure (in the sense of a power-law behaviour). It might be surprising, then, that the generalised radial flow model and fractal theory are intrinsically related and in fact almost equivalent. This might not at first be obvious but it is useful to point out here that Barker's model implicitly made use of a generalised diffusion equation, which is based on Brownian motion. Brownian motion has been shown before to exhibit fractal properties (Mandelbrot, 1977). For example, ground-water flow is generally described by a diffusion equation such as $\frac{\partial h}{\partial t} = D\nabla^2 h$, where h is the hydraulic head and D a diffusion coefficient. A random walker, as in Brownian motion, is also diffusing. A first hint that Barker's model is related to a spatial structure of the underlying flow can be seen from the dependency of the flow dimension on the non-space-filling properties of the medium as discussed above. A discussion of fractal theory is given below. The reader familiar with the basics of fractal theory and different ways of how to measure a fractal dimension might omit this section, even though it will be instructive to follow the line of argument in order to appreciate why fractal theory is such a powerful tool in hydrogeology (and not just for pumping test analysis). A later section shows how fractal theory has been used to develop a mathematical description of a general flow equation, assuming that the flow medium has an underlying fractal structure.

2.3.1 Definition of fractals

A fractal, or a fractal set is "a rough or fragmented geometric shape that can be subdivided in parts, each of which is (at least approximately) a reduced-size copy of the whole" (Mandelbrot, 1977). A fractal set can be characterised by its "fractal dimension" which is an indication for the heterogeneity or roughness of the shape. Mandelbrot also defined a fractal to be a set with Hausdorff dimension[2] strictly larger than its topological dimension. Fractal systems have a self-similar or self affine geometry at different measurement scales. This means that measurements made at one scale can be used to predict geometries at other scales (*scale independence*). This is one of the reasons why fractals have been extensively studied in nature. Since much of geology and hydrogeology is related to the geometry and topology of geological features such as faults, fractures, lithology and hence pore size distribution, many researchers have proposed the use of fractal dimension as an index for replicating geological realistic fracture patterns and to extrapolate features from a measured scale to a different scale. Even though the term "fractal" has become a very popular term, it simply describes that a scale independent property can be related to another by a power law equation as explained next.

[2] The Hausdorff dimension is defined as the value of d for which the measure $M_d = N(L) \cdot L^d$ is finite and non-zero, with $N(L) \propto L^{-D}$, where M is a "mass", N a count, L a length and D a dimensional exponent. For Euclidean objects, D is an integer, so the measure is only finite and non-zero when d equals its topological dimension, whereas for fractals d is always a non-integer. For the original definition see Edga (1993).

2.3.2 Fractal geometry

Most geometric forms used for building man-made objects belong to Euclidean geometry, they are comprised of lines, planes, rectangular volumes, arcs, cylinders, spheres, etc. These elements can be classified as belonging to an integer dimension, either 1, 2, or 3. This concept of dimension can be described intuitively and is trivial as a mathematical description. Intuitively it is also clear that a line is one dimensional because it only takes one number to uniquely define any point on it with respect to a given origin. That one number could be the distance from the start of the line. This applies equally well to the circumference of a circle, a curve, or the boundary of any object.

Figure 8: Any point "a" on a one dimensional curve can be represented by one number, the distance d from the reference point.

A plane is two dimensional since in order to uniquely define any point on its surface two numbers are required. Typically, these two numbers are defined from an orthogonal co-ordinate system. Other examples of two dimensional objects are the surface of a sphere or an arbitrary twisted plane.

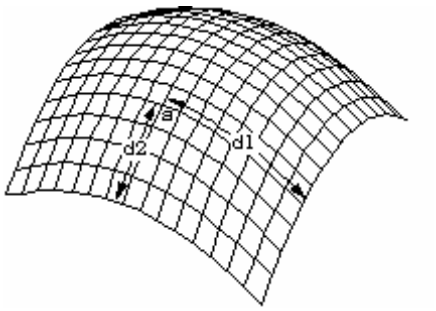


Figure 9: Any point "a" on a two dimensional surface can be uniquely represented by two numbers. One of the many possible methods is to grid the surface and measure two distances along the grid line.

The volume of some solid object is three dimensional on the same basis as above: it takes three numbers to uniquely define any point within the object.

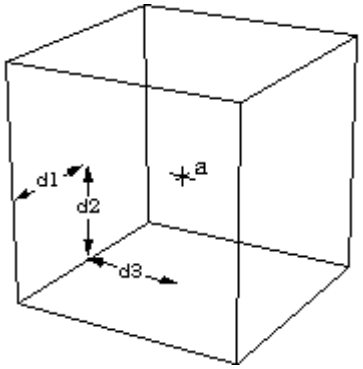


Figure 10: Any point "a" in three dimensions can be uniquely represented by three numbers. Typically these three numbers are the co-ordinates of the point using an orthogonal, Euclidean, co-ordinate system.

A more mathematical description of dimension is based on how the "size" of an object behaves as the linear dimension increases. Consider a one dimensional line segment.
If the linear dimension of the line segment is doubled then obviously the length (characteristic size) of the line has doubled also. In two dimensions, if the linear dimensions of a rectangle is doubled then the characteristic size, the area, increases by a factor of 2^2. In three dimensions if the linear dimension of a box are doubled then the volume increases by a factor of 2^3.

This relationship between dimension D, linear scaling of L and the resulting increase in size S can be generalised and written as $S=L^D$. This is just expressing mathematically what is known from everyday experience. If, for example, a two dimensional object is scaled, then the area increases by the square of the scaling. If a three dimensional object is scaled, the volume increases by the cube of the scaling factor.

Rearranging the above gives an expression for dimension depending on how the size changes as a function of linear scaling, namely

$$D = \frac{\log S}{\log L} \quad \text{.....(21)}$$

In the examples above the value of D is either 1, 2, or 3, depending on the dimension of the geometry. This relationship holds for all Euclidean shapes.

The dimension as defined by degrees of freedom looks natural. However, it contains a serious flaw. Nearly one hundred years ago people like Peano, Koch and Sierpinski described lines that are space filling, non-differentiable and of infinite length (Takayasu, 1989).
Hence, there are shapes which do not conform to the integer based idea of dimension given above by both, the intuitive and the mathematical descriptions. There are objects which have a topological dimension of one, i.e. a line, but for which a point on the curve cannot be uniquely described with just one number. These curves are partially *space filling*. If the earlier scaling formulation for dimension is applied, the formula does not yield an integer. There are shapes that lie in a plane but if they are linearly scaled by a factor L, the area does not increase by L squared but by some non integer amount. These geometries are called “fractals” (Mandelbrot, 1977). One of the simpler fractal shapes is the Koch curve, or “snowflake”. The method of creating this shape is to repeatedly replace each line segment with four line segments:

The first few iterations of this procedure are shown in the following figure. The process starts with a single line segment and continues over an infinite number of scales.

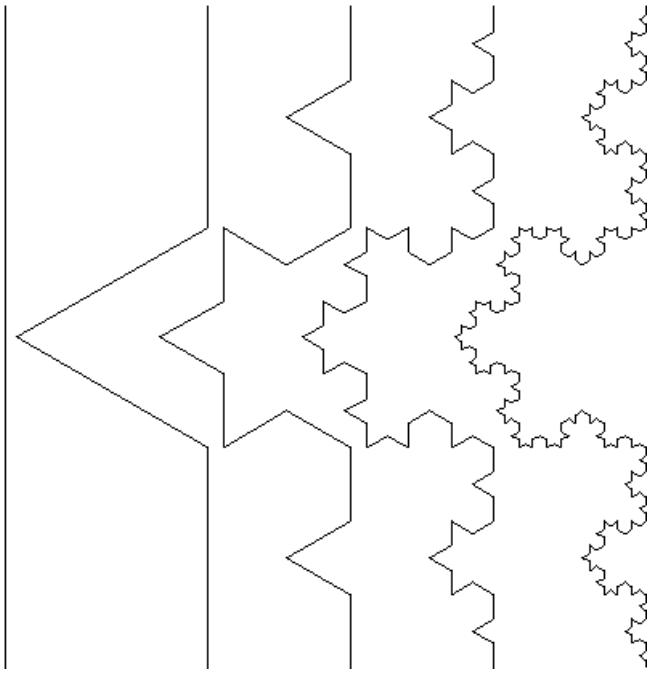


Figure 11: The construction of the Koch snowflake or curve. This shape has a fractal dimension of D=log4 / log3.

This demonstrates how a very simple generation rule for this shape can generate some unusual (fractal) properties. Unlike Euclidean shapes this object has detail at all levels. If one magnifies an Euclidean shape such as the circumference of a circle it becomes a different shape, namely a straight line. If, however, a fractal as in figure 12 is magnified further, more and more detail is uncovered and the detail is exactly self similar, where any magnified portion is identical to any other magnified portion.

Note also that the "curve" on the right of figure 7 is only an approximation of a fractal. This is similar to the fact that drawing a circle is only an approximation to a perfect circle.

At each iteration the length of the curve increases by a factor of 4/3. Thus the limiting curve is of infinite length and indeed the length between any two points of the curve is infinite. The Koch curve manages to compress an infinite length into a finite area of the plane without intersecting itself. Considering the intuitive notion of one dimensional shapes, although this object appears to be a curve with one starting point and one end point, it is not possible to uniquely specify any position along the curve with one number as we expect to be able to do

with Euclidean curves which are one-dimensional.

2.3.3 Self-similar, self-affine and statistically self-similar fractals

An important concept in fractal geometry is that of self-similar and self-affine fractals. Self-similar shapes possess only exactly one scaling factor, whereas self-affine shapes show a different scaling in different directions. This difference in scaling can lead to an anisotropic behaviour. The difference between self-similar and self-affine sets is illustrated in figure 13.

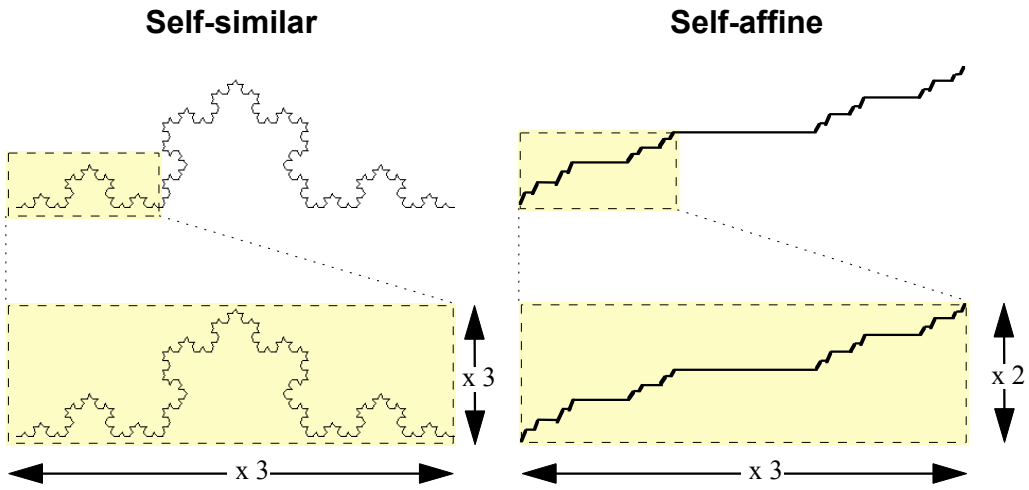


Figure 13: The triadic Koch curve on the left is self-similar with the same scaling factor along axes. The "Devil's staircase" on the right shows two different scaling factors and is hence self-affine. (After Mandelbrot, 1977)

The original definition of fractals is restricted to self-similar shapes, even though self-affine structures are probably more common in nature than self-similar shapes. This is especially true for fault traces (Dershowitz et al., 1992).

Yet another category of fractals can be called *statistically self-similar*. This is exemplified by Brownian motion projected onto a plane as shown in figure 14. The path of a particle is statistically the same, independent on magnification. With a bounded random set, statistical

self-similarity is achieved because the distribution of any particle in Brownian motion is identical to the distribution of all particles (Mandelbrot, 1977).

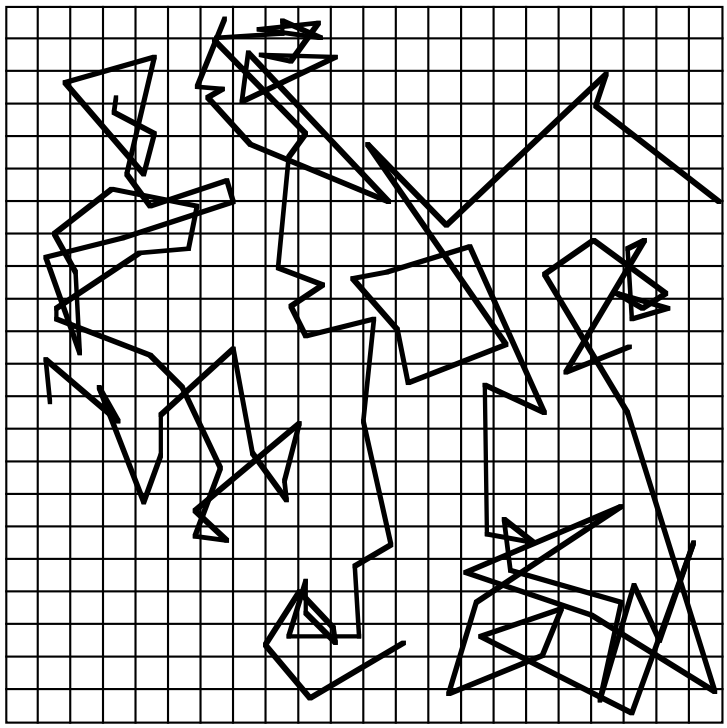


Figure 14: Brownian motion of several particles in s 2-D plane. Even though individual paths are not similar, the statistical properties are the same. These types of shape are called statistically self-similar. (After Mandelbrot, 1977)

An important point to notice is that estimation methods for fractal dimension might only be applicable in a mutually exclusive way to either similarity concept. Often, of course, the exact nature of self-similarity or affinity will not be known, so a method should be chosen that can be applied to both scaling types.

2.3.4 How to measure fractal dimension

More than twenty different approaches of estimating the fractal dimension of an object have been described (Dershowitz et al., 1992). It is possible to derive many numerically different fractal dimensions, each one relating to a different aspect of the underlying object. It is thus important to always state the measurement method used, otherwise a fractal dimension becomes meaningless. Fractal dimension measures are generally based upon a power-law relationship between a size measure and a count measure, with the fractal dimension defined by the slope of a *log-log plot* as shown in figure 15. This directly follows from equation 21.

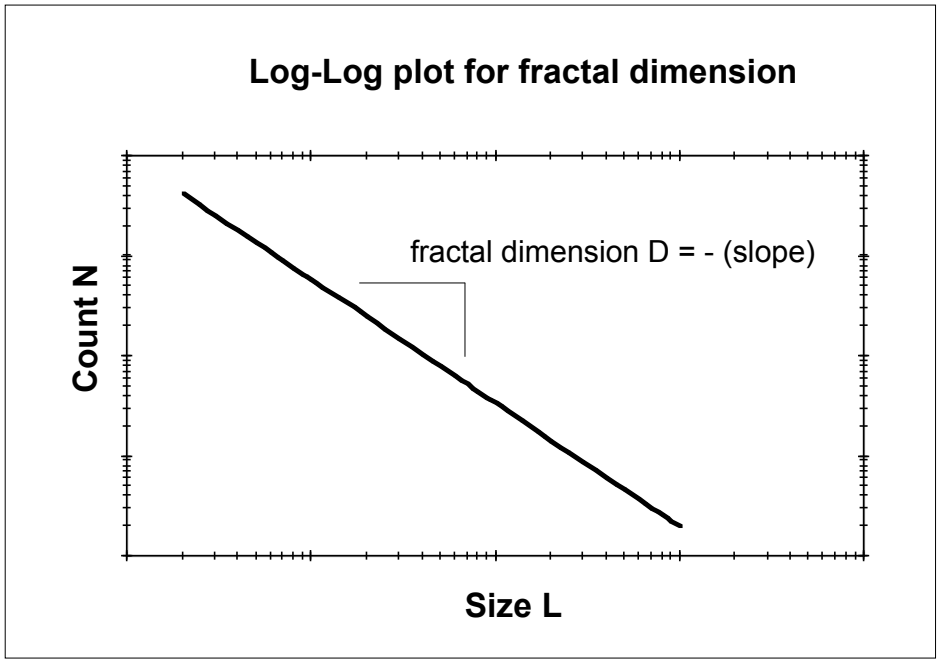
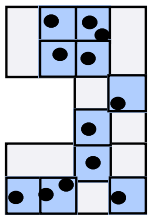


Figure 15: This figure illustrates how a fractal dimension is generally obtained from the negative slope of a *log-log plot* of a count measure vs. a size measure.

Several methods that can be used to estimate fractal dimensions are listed in table 1, according to whether they are applicable to self-similar or self-affine fractals. They can be classified according to how the count measure N and the "size" measure L, which is equivalent to a "ruler length", are defined.

Self-similar methods

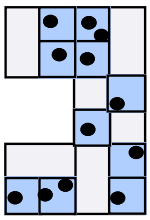
a) Box dimension



Count N:
Number of boxes needed to cover features. Here two count generations are shown

Length L:
Size of boxes

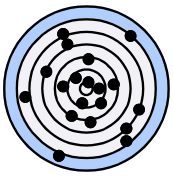
b) Information dimension



Count N:
Weighted number of boxes needed to cover features

Length L:
Size of boxes

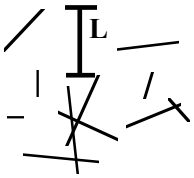
c) Mass/ Cluster/ Density dimension *



Count N:
Number of features inside circle

Length L:
Radius of circle

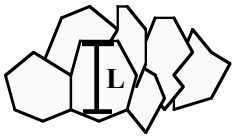
d) Correlation dimension



Count N:
Number of features less than L distance apart

Length L:
Distance between features

e) Correlation dimension

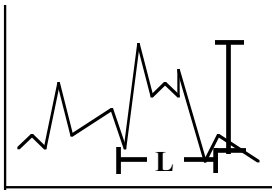


Count N:
Number of features

Length L:
Max. distance inside blocks

Self-affine methods

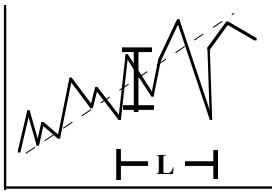
a) Rescaled range



Count N:
Normalised maximum difference in values

Length L:
Interval length

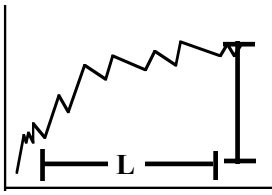
b) Root-mean-square roughness (RMS)



Count N:
Normalised RMS error from trend

Length L:
Interval length

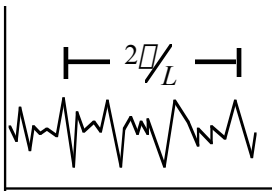
c) Variogram analysis



Count N:
Normalised difference between points

Length L:
Distance between points

d) Spectral density



Count N:
Depth of power (frequency) spectrum

Length L:
Wave number

* The radial mass/ cluster/ density dimension method is also suitable for self-affine sets.

Table 1: This table shows several methods that can be used to obtain a fractal dimension for self-similar and self-affine sets. Methods can be categorised by the how the count measure, N, and the "size" measure L are defined. (After Dershowitz et al., 1992)

Examples of the practical application of selected methods mentioned before are given next:

The divider method

The divider method is implemented by placing a minimum number of yardsticks of fixed common intervals over a structure so that it is fully covered. The number of intervals required is counted and multiplied by the length of the interval to yield a fractal length, L. Then the interval is shortened and the process repeated. The fractal length is then related to the fractal dimension D and the interval length r by $\log L = A + (1 - D)\log r$, where A is the y-axis intercept.

The box-counting method (which defines the "capacity dimension")

The *box-counting method* is essentially a two-dimensional yardstick. The procedure for obtaining a fractal dimension is as follows: The minimum number of square boxes of a certain size L that are needed to fully cover every feature of the graph is determined and then plotted on a log-log graph of N vs. L. The box size is then halved and the procedure repeated. The fractal "capacity" dimension is defined by the negative slope of a best-fitting straight line through this graph. This method is also applicable to a three-dimensional Euclidean embedding space.

A dimension determined in this way has two main disadvantages:

- It does not account for the *frequency* with which the set in question might "visit" the covering cells, and thus local properties of the set properties pertaining to neighbourhoods of individual points are not distinguishable.
- Calculating truly minimal coverings of a set is computationally intensive.

These disadvantages are overcome by defining the *information dimension.*

The information dimension method

Let $\|S\|$ denote the number of elements in a set of points, S, and for each of the N(e) cells, where e is the size of the side of a cell, let N(e,i) be the number of points in cell number i. Set $P(e,i) = \frac{N(e,i)}{\|S\|}$ the sample probability that a point of S is in the ith cell. I(e) measures the average surprise in learning of which cell a point is in. The quantity $-\log(P(e,i))$ measures the information conveyed by knowing that a point of S is in the ith cell of the covering, and $I(e) = -\sum_{i=1}^{N(e)} P(e,i)\log(P(e,i))$ measures the *average* information conveyed by knowing what cell a point of S is in. Now note that when all the probabilities are equal, each P(e,i) is equal to 1/N(e), and the sum above is just I(e)=log[N(e)]. This is, in fact, the maximum possible value of I(e). The lower values that might occur can be interpreted as quantifying the non-uniformity of the distribution of the points, or alternatively, as correcting the capacity dimension estimate by giving less weight to the cells that contain relatively few points. Thus, if a fractal dimension is defined by: $D = \lim_{e\to 0} \frac{I(e)}{\log(\frac{1}{e})}$, a formulation of fractal dimension is obtained that is similar to the capacity dimension but which gives a value different from the capacity dimension when the distribution of the set over the covering is non-uniform.

The correlation dimension method

Correlation dimension can be calculated using the distances between each pair of $\|S\|$ points in the set S: $s(i,j) = \|\mathbf{x}(i) - \mathbf{x}(j)\|$. This gives $\|S\|^2$ values, and it is possible to calculate a sample correlation function C(r) with the formula $C(r) = \|S\|^{-2} x$ (number of pairs (i,j) with s(i,j) < r). The function C(r) has been found to exhibit a power law like the one obtained for capacity dimension: $C(r) = kr^D$; and we can aim to find D with estimation techniques derived from the formula: $D = \lim_{r\to 0} \left|\frac{\log C(r)}{\log r}\right|$.

The Mass/ Cluster/ Point method

This method is similar to the box-counting method but can be used for both, self-similar and self-affine sets. Here a series of circles is defined, starting at the centre of a cluster of fractures. The number of features inside each circle is then plotted against the circle radius, giving a power-law relationship of the form $\square(r) = r^D$. Note that this method can be modified such that instead of counting points inside a given circle, the total fracture volume or fracture intersection per volume are plotted against the radius. An extension of this method has been used for this study (see section 4.7).

The spectral density method

This method assumes that the fractal has a spectral density that follows a power-law relationship. In other words, the spectral density of frequency *f* is proportional to f^{-b} and the spectral dimension D is related to b as b=2D-5 (Cox and Wang, 1993). This method is restricted to self-affine sets.

Other fractal dimension methods

Other fractal dimension definitions include:

- Slit-island method: In this technique, the topographical surface of a function is sliced horizontally, creating “islands” and “water”. Perimeter and area are then determined for each island individually and plotted as perimeter vs. area. The slope of this log-log plot then gives 2/D where D is the fractal dimension. Mandelbrot (1977) addresses problems that can arise if there are “lakes within islands” or “islands within lakes”. This method can be directly applied to surfaces, whereas most other methods are designed for curves and have to be adapted to measure fractal dimensions of fractal surfaces.
- Fragmentation dimension: Based on the distribution of block sizes as measured by the longest ruler which would fit into each block.

2.3.5 Fractal dimensions for different Euclidean embedding dimensions

Several problems with the application of fractal dimensions exist. Two basic ones will be explored in this and the following section.

Generally, it is not possible to infer the fractal dimension of a 3-D fracture network from a 2-D network or from planar section through a 3-D network. This problem is illustrated in figure 16, where a) shows a modified Sierpinski gasket proposed by Sammis (1987) to model fragmented rock found in fault zones. Here, the adding of one embedding dimension simply increases the fractal capacity dimension (D_0) by one. However, in the general case, D_0(3-D)=D_0(2-D) +1 is only true if D_0(2-D) is the spatial average of all possible sections. For example, if the fractal dimension of a Sierpinski Triangle is measured in a 2-D projection, then D_0(2-D)=1.58, but the entire 3-D pyramid gives D_0(3-D)=2.0. In figure 16a, all 2-D sections parallel to a face have the same fractal dimension. In figures 16b and 16c, the 2-D fractal dimension varies with the orientation of the section.

Interpreting this observation with respect to a random (natural) fractal suggests that simple up-scaling of fractal dimension with embedding dimension only holds if the structure is homogeneous, i.e. where all 2-D sections have the same fractal dimension.

As a result, the mapping of fractures and faults in 2-D sections (e.g. outcrop mapping and in boreholes) does not normally allow the direct estimation of fractal properties in 3-D.

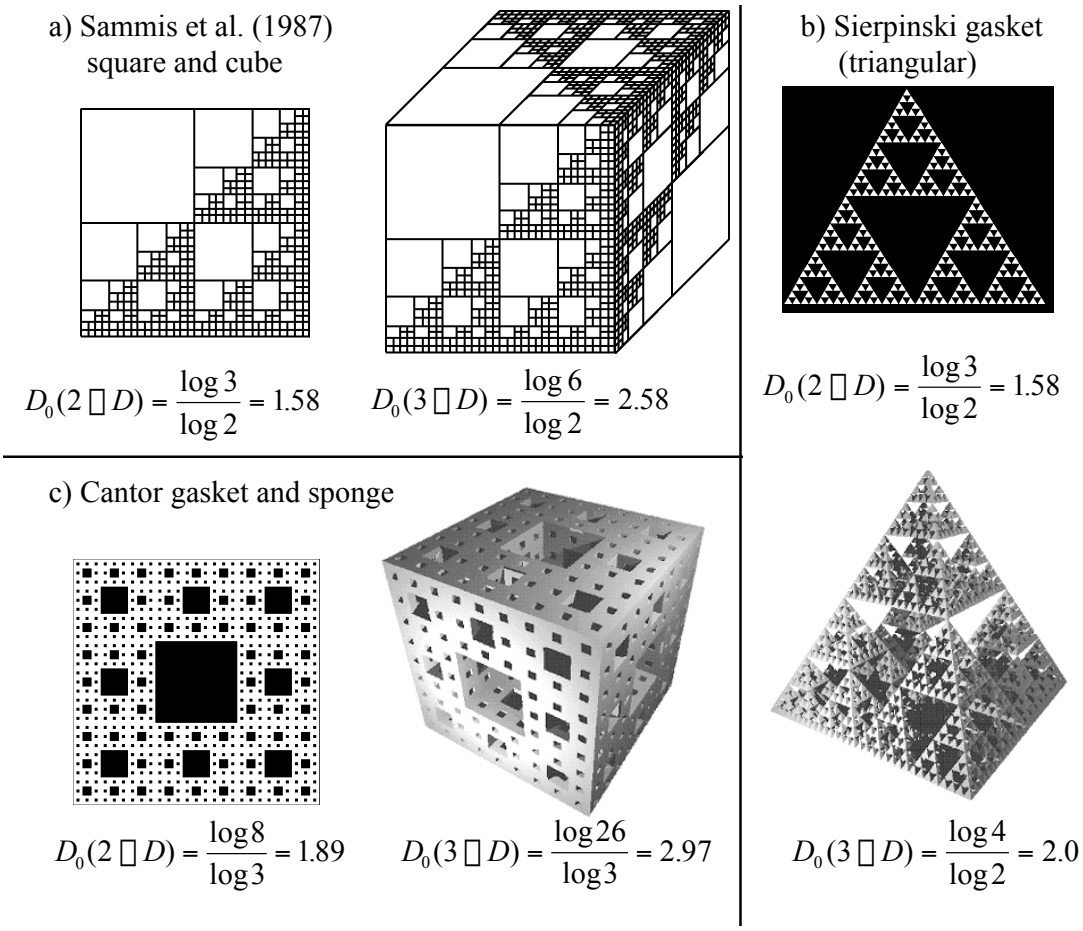


Figure 16: a) Two- and three-dimensional illustrations of the idealised fractal structure (modified Sierpinski gasket) for fault zones proposed by Sammis et al. (1987). Note that the 3-D version shows a fractal capacity dimension equal to the 2-D version plus one. b) The 3-D Sierpinski pyramid has a fractal dimension that is only 0.42 higher than the 2-D gasket. c) The fractal dimension of the 3-D Menger Sponge (see Mandelbrot, 1977) is 1.08 higher than that of the Cantor gasket.

2.3.6 The problem of multi-fractals and lower/upper cut-off length scales

It is clear that the structure of many materials in nature, especially if they are fractured, is not the result of just one process. Instead, a sequence of several processes will act to shape, modify and characterise the properties of any rock mass. Examples include several phases of tectonic deformation, mineral dissolution and precipitation, diagenesis and sedimentary reworking etc. It will be shown in a later section that the fractal properties of fractured rock are related to the governing fracturing mechanism. Hence it becomes clear that several fractal structures might be superimposed in natural rocks. This leads to the problem of *multi-fractal* structures.

In the strict mathematical sense, multi-fractal structures are defined by their *local* fractal dimension, which might not be a constant (Takayasu, 1989). Space-varying fractal dimensions are related to a cascading frequency spectrum and will not be considered further in this study. Other researchers have used the term "multi-fractal" for log-log plots that can be approximated by more than one straight line. Then the structure has a constant fractal dimension only over a certain length range. This leads to the problem of cut-off length scales. As described in section 2.3.1 , the fractal dimension is defined over an infinite number of length scales. In nature, however, this is only approximated: There are physical limits on the smallest and the largest scale over which structures can be self-similar. For example, Garrison et al. (1993a,b) found that the smallest pore-size generation observed in thin-sections do not seem to contribute very much to measured core-plug air permeabilities and a better fitting fractal exponent was obtained when the first pore-size generation was omitted.

Even though Karasaki et al. (1988) found that very small fractures might not be important for modelling flow in fractured media, and might be approximated by an equivalent porous block approach, Acuna and Yortsos (1995) report that numerical simulation of flow through fractal networks is only similar to the analytical solution if the object is fractal over a large enough range of scales. Otherwise, finite size and storage effects may dominate the pressure response and make the identification of the structure difficult. This emphasises the importance of considering finite length scales that are either imposed my computational constraints in the case of synthetic networks or by physical processes for real aquifers.

2.4 Review of fractal structures in hydrogeology

2.4.1 Introduction

Fractal dimension measurements have been reported by many researchers in the field of earth sciences. This section aims to demonstrate that fractal theory is not only a good method to describe a power-law behaviour between different parameters (as a mathematical model approximation), but also that fractal structures seem to intrinsically govern many of the processes that are important in hydrogeology. Applications of fractal measurements in the earth sciences can be categorised into the following fields:

- testing whether some feature shows fractal scaling
- characterisation of surface geometry to determine some internal property
- use of fractal geometry to study formation and degradation processes
- use of fractal slopes to determine multiple processes and the scales over which they are dominant
- use of fractal geometry as a tool for interpolation and extrapolation
- use of fractal geometry to derive empirical equations to estimate parameters that are difficult to measure.

A good review of fractal applications in the earth sciences was given by Takayasu (1989) and Cox and Wang (1993). Table 2, modified after Cox and Wang (1993), shows applications and methods used. This review shall focus on applications of fractal measures on properties that are of direct interest for hydrogeological investigations and may directly determine apparent diffusivity and connectivity, such as

- pore geometry (aggregate, particle and void size distribution)
- fracture surfaces (roughness)
- fault traces (surface mapping and borehole intersections)
- fracture networks
- others (river geometry, Karsts etc.)

Method	*Reference*	*Application*	*E - D*
Divider	Norton et al., 1989	Granite mountain profile	.15 to .28
Divider	Snow, 1989	Stream channels	.04 to .38
Divider	Aviles et al., 1987	San Andreas Fault trace	.0008 to .0191
Divider	Brown, 1987	Rock fracture surface	.50
Divider	Carr, 1989	Rock fracture surface	.0000 to .0315
Divider	Miller et al., 1990	Rock fracture surface	.058 to .261
Divider	Underwood et al., 1986	Steel fracture	.351 to .512
Divider	Akbarieh et al., 1989	Erosion of Ca-oxal, crystals	.025 to .106
Divider	Kaye, 1986	Carbon particles	.32
Divider	Kaye, 1986	unpolished Cu surface	.47
Divider	Kaye, 1986	polished Cu surface	.00
Box	Barton and Larsen, 1985	Rock fracture network	.12 to .16
Box	La Pointe, 1988	Rock fracture network	.37 to .69
Box	Miller et al., 1990	Rock fracture network	.041 to .159
Box	Hirata, 1989	Japan fault network	.05 to .6
Box	Okuba and Aki, 1987	San Andreas Fault trace	.2 to .4
Box	Sreenivasan et al., 1989	Turbulent flow interface	.35
Box	Langford et al., 1989	Epoxy fracture	.35
Box	Langford et al., 1989	MgO fracture	.16
modified Box	Garrison et al., 1993	carbonates and sandstone	empirical eqn. K & E
Triangle	Denley, 1990	Gold film surface	.04 to .46
Slit-island	Mecholsky and Mackin, 1988	Chert fracture	.12 to .32
Slit-island	Schlueter et al., 1991	Sandstone pores	.31 to .40
Slit-island	Schlueter et al., 1991	Limestone pores	.20
Slit-island	Huang et al., 1990	Steel fracture surface (lakes)	.20 to .30
Slit-island	Huang et al., 1990	Steel fracture surface (islands)	.33 to .40
Slit-island	Mandelbrot et al, 1984	Steel fracture surface	.28
Slit-island	Pande et al., 1987	Titanium fracture surface	.32
Slit-island	Langford et al., 1989	Epoxy fracture surface	.32
Spectral	Gilbert, 1988	Sierra Nevada topography	-0.835 to .471
Spectral	Brown and Scholz, 1985	Rock fracture	.26 to .68
Spectral	Carr, 1989	Rock fracture	-0.880 to .467
Spectral	Miller et al., 1990	Rock fracture	.124 to .383
Spectral	Mandelbrot et al., 1984	Steel fracture	.26
Spectral	Langford et al., 1989	Photon emiss. from epoxy fract.	.45
Variogram	Burrough, 1983	Soil pH variation	.6 to .8
Variogram	Burrough, 1983	Soil Na variation	.7 to .9
Variogram	Burrough, 1983	Soil elec. resist. variation	.4 to .6
Variogram	Armstrong, 1986	Soil microtopography	.64 to .90
Distribution	Curl, 1986	Cave length, volume	.4, .8
Distribution	Krohn, 1988a	Sandstone pores	.49 to .89
Distribution	Katz and Thompson, 1985	Sandstone pores	.57 to .87
Distribution	Krohn, 1988b	Carbonate and shale pores	.27 to .75
Distribution	Avnir et al., 1985	Carbonate particles	.01 to .97
Distribution	Avnir et al., 1985	Soil particles	.43 to .99

Table 2: Measured fractal dimension for different geological applications. The right column denotes the difference between the Euclidean embedding dimension E and the fractal dimension D as determined by the method indicated in the left column. Applications relevant to Hydrogeology are highlighted.

(Modified after Cox and Wang, 1993)

2.4.2 Pore geometry

It is well known from Poiseuille's law that the rate at which a fluid can travel through a porous medium is controlled by the flow paths that are open to flow. This flow path is a subset of the overall geometry of the pore system. Garrison et al. (1992, 1993) showed that the apparent surface fractal dimension D_S, obtained from the pore diameter vs. number distribution, exhibits a wide range of values for any particular rock type. This distribution contains information about lacunarity and multiple fractal processes. They showed that there is an empirical relationship between measured core plug air permeability, shape factor and pore similarity fractal dimension D_S, which is valid for several different reservoir rock types. Figure 17 shows that only the fractal similarity dimension D_S and the shape factor S_a are needed to account for all of the variability in measured core plug permeability. This is obviously of great use in situations where no core can be taken and only rock chippings or fragments are available because only a small thin section of the rock is needed to measure the necessary parameters.

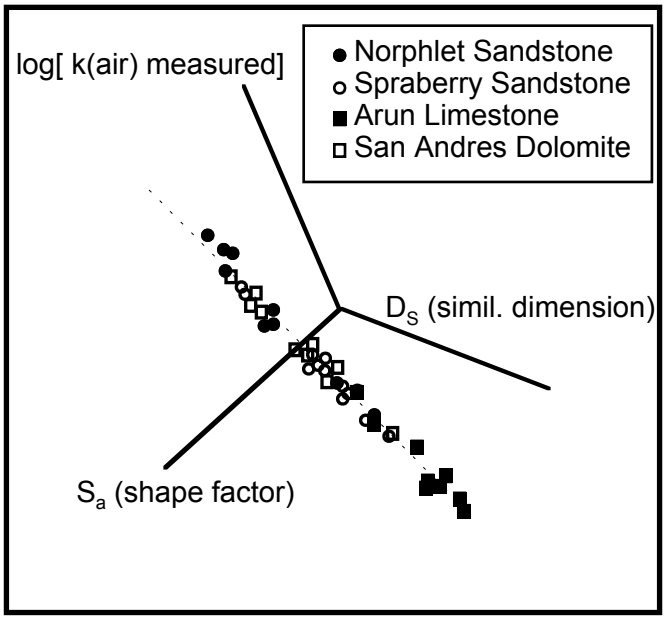


Figure 17: A 3-D plot, in log(k)-D_S-S_a space, of measured core plug air permeability vs. apparent similarity dimension D_S and area shape factor S_a for the pores of the controlling process in samples of various reservoir rocks. Note that the smallest pores are excluded. (After Garrison et al., 1993c)

Researchers that have used fractal dimensions to classify pore geometries include Katz and Thompson (1985), Krohn and Thompson (1986) and others (see review by Thompson, 1991). The method used usually is to analyse the fractal geometry of the pore structure of sedimentary rocks by imaging impregnated thin sections and counting the number of pores for different diameters. Avnir et al. (1985) re-analysed existing particle distribution data of particle aggregates of different origin (carbonates, quartz-, rock- and soil particles). They found distinct fractal dimensions for carbonate rocks of different origin.

2.4.3 Fracture surfaces

Fracture surfaces have been the subject of intense study in connection with rock characterisation for subsurface nuclear waste repositories. This interest was motivated by the observation that flow is often observed to be channelled along fracture planes rather than flowing across the whole fracture plane. The amount of this channelling is controlled by the roughness of fracture surfaces.

Both, field and laboratory rock fractures have been analysed by several researchers. In the field, natural fractures that formed after the disruption of mineral grains, rock fragmentation and matrix filling are subjected to further movement along the fault and filling of the fracture by fluid flow interaction. This complicates the interpretation of natural fracture surface topography.

Brown and Scholz (1985) measured parallel sets of profiles across both laboratory and field-scale rock surfaces. They included different fracture types such as bedding planes, fault planes and glacial surfaces. The measured fractal dimension was then used to generate rough surfaces (figure 18) and to evaluate fluid flow on fractures composed of rough surfaces. Fluid flow was then simulated between these surfaces. Brown and Scholz concluded that fractal dimension was constant only over limited ranges of spatial frequency and also that the fractal description of fractal surfaces offers an advantage over topographic measurements because other roughness measures are not constant over different scales. However, it was also found that variations of fractal dimension produce only second order effects on the fluid flow.

Carr (1989) evaluated the effect of using different fractal dimension techniques (spectral dimension and divider method) on the correlation with a joint roughness coefficient. They

found that the divider method is most suitable to correlate the fractal dimension of the Yucca Mountain fracture surfaces with the roughness coefficient.

Miller (1990), on the other hand, found that the correlation between fractal dimension and roughness of fracture surfaces is poor and that the main use of fractal dimension characterisation is the ability to support simulations and visualisations rather than as a means to uniquely describe fracture roughness.

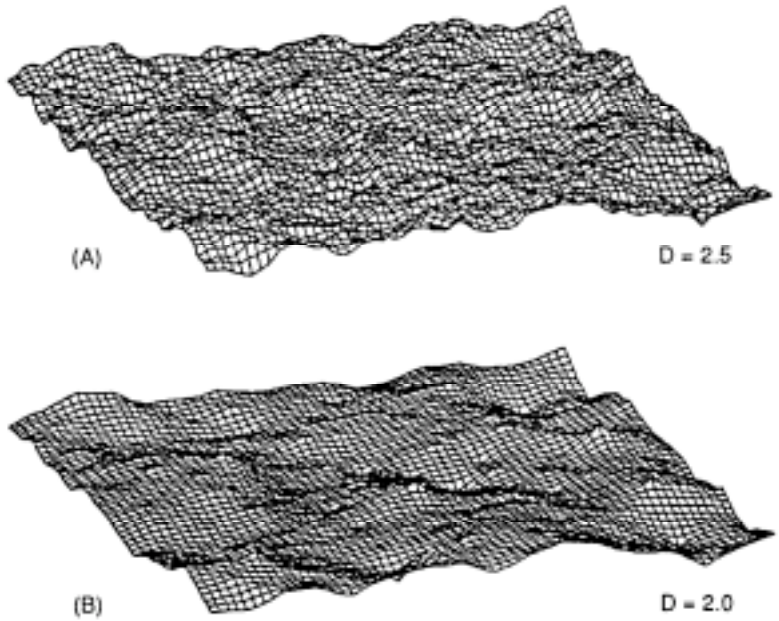


Figure 18: Examples of fractal surfaces used in the simulations. Both surfaces have the same root-mean-square height. Both were generated with the same set of random numbers but with different fractal dimensions: A) 2.5 and B) 2.0.
(After Brown, 1989)

2.4.4 Mapping of fault traces and lineaments

The mapping of fault traces and lineaments is probably the field with the most direct applicability to fractured rock hydrogeology. Many publications have reported the measurement of fractal dimensions obtained from fault traces at the surface and in boreholes. Scholz and Aviles (1986) digitised fault traces from the San Andreas Fault and found a spectral fractal dimension ranging from 1.1 to 1.5. Okuba and Aki (1987) used the box counting method on traces from the San Andreas area, trying to relate strain release to the geometry of traces. Circles of different radii were used to cover the faults. Fractal dimensions

obtained in this way were between 1.12 and 1.43. Aviles (1987) used the divider method and obtained a fractal dimension close to one.

The analysis of fault traces has also been extended to networks of branching fracture traces. Hirata (1989) reports fractal analyses using the box counting method on fault systems in Japan. One aim was to determine whether the structure of the fault system was self-similar. The fractal dimensions obtained were between 1.05 and 1.60, with high values observed at the centre of the structures, decreasing away from the centre of the Japan Arc.

Barton and Larsen (1985) and La Pointe (1988) measured the fractal dimensions of fracture networks in rock pavements for areas ranging from 200 to 300m^2. Barton and Larsen found that for Miocene ash-flow tuff pavements at the Yucca Mountain Range fractal dimensions range from 1.10 to 1.18. The fracture trace length followed a log normal distribution. A sub-analysis of the same data showed that, even though the fracture networks appeared visually different, the fractal dimension of different networks was similar (1.12, 1.14 and 1.16).

La Pointe (1988) developed an index of fracture density based upon fractal dimension for two dimensional visualisations of fractures. He used two approaches, in one of them the number of fractures per unit area of rock is counted for each different grid spacing. The second approach counts the number of blocks bounded by fractures in each grid. It was found that fracture density, as determined by either method, was fractal and scale invariant. The study also concluded that the block size distributions may be controlled significantly by the fractal dimension of the geology.

2.4.5 Fractal relationship between fault number, lengths and widths

Watanabe and Takahashi (1995) have given a very good example of the usefulness of the fractal characterisation of fault traces. They researched the properties of fractured rocks with respect to their use as geothermal reservoirs. Instead of using the box counting method, which gives a measure for the spatial distribution of fractures, they use a power-law relationship between observed fracture length r and the number of fractures N whose length is equal to or larger than this length. This can be expressed by the equation $N = Cr^{-D}$, where C is a fracture density parameter. This study also compared fractal networks simulated by using the power-law relationship with percolation models and it was found that the number of fractures needed

to obtain flow is significantly lower than that of percolation models. This study demonstrated that it is possible to characterise a subsurface fracture network with a few parameters obtained from borehole data.

Main, Meredith et al. (1990) reported a value for the fractal dimension measured in this way of 1.3. This relationship is related to the observation that earthquake fault length, slip (and therefore magnitude) and frequency also scale as a power-law (Scholz and Cowie, 1990). Hence, this is strong (albeit circumstantial) evidence that the fractal relationship observed between different fault parameters are caused by the underlying fracturing mechanisms.

A similar relationship has been reported by Thomas and Blin-Lacroix (1989) for fractures with width w and the number of fractures whose widths are equal to or larger than w. Since fracture aperture and length directly determine the hydraulic conductivity of a given fracture, a fractional flow dimension can also be expected.

2.4.6 Fractal distribution of three-dimensional fracture networks

Robertson and Sammis (1995) have reported a novel approach to the fractal analysis of fracture networks. Although many studies that were reviewed above have demonstrated that faults and fractures are self-similar over a large range of scales in two dimensions, none have attempted to extend a fractal analysis to three dimensions. It is, of course, difficult to obtain a good representation of the orientation of fractures in three dimensions with conventional methods. Robertson and Sammis, however, used earthquake hypocentral locations in central and southern California to “illuminate” three-dimensional fault structures, for which the three dimensional capacity dimension D_0(3-D) was obtained. Aftershocks and background seismic events were found to show a fractal pattern, asymptotically approaching a constant dimension value for a large number of events. D_0(3-D) dimensions reported for three different earthquake zones were 1.92±0.02, 1.82 and 1.79. As has been shown in section 2.3.5, it is not always possible to infer the fractal capacity dimension of a 3-D fracture network from a 2-D planar section. Nevertheless, the values reported here are not much higher than the 2-D values reported in the section above, even though they could be expected to be from the example of homogeneous fractal structures, where adding one embedding dimension also increases the fractal dimension by one (see section 2.3.5). The authors interpreted this finding such that

earthquakes only occur on the "percolation *backbone*" of a fault network, i.e. the active part of the network that accommodates finite strain deformation (Sahimi et al., 1993). The authors also showed that a percolation model that allows for healing of previously broken bonds is consistent with this interpretation. This might be linked to the important part that water plays in altering the physical properties of rock.

2.4.7 Other applications of fractal analysis

It should be briefly noted here that there are many other applications of fractal analysis mainly falling into the fields of topography, river and stream geometry, Karsts and other geological processes. Even though these are not directly important from a hydrogeological point of view, these processes might still govern properties like spatial pore size distribution variations, e.g. a fractal river geometry might lead to a fractal hydraulic conductivity distribution.

Curl (1986) measured the fractal dimension of caves and found D=1.4 for the length distribution and 2.8 for the volume, notably the same as the already mentioned Menger sponge (see section 2.3.5 and also Garrison et al., 1993).

Snow (1989) applied fractal analysis to stream channels and related the fractal dimension to their sinuosity. Using the divider method, he found values ranging from 1.04 to 1.38.

Many theoretical studies of fractal structures have also been published. Adler (1985) examined the problem of Taylor dispersion in capillary networks exhibiting fractal behaviour. It was concluded that the larger the network, the longer any tracer particles are retained. This was explained by the regular presence of "bottlenecks" that alter the large-time behaviour of the various moments. The result has obvious implications for the analysis of tracer tests and for contaminant transport problems and it will be shown in a later section on fractal flow theory that this principle of "bottlenecks" is also exhibited by flow in fracture networks.

Fractal structures have also been related to the fracture resistance of materials and to the particular fracturing process (Turcotte, 1986). Fragmentation with substantial shearing, which appears to be a dominant mechanism for many fracture networks, leads to fractal dimension values ranging between 1.2 and 1.8 (Brown et al., 1983).

2.5 Pressure-transient analysis of fractal reservoirs

2.5.1 Introduction

Several workers have now extended the analysis of pressure-transient flow to fractal media. It has already been shown how Barker (1988) developed a model for radial flow where the cross-sectional *flow-area* scales with non-integer exponents. It was also pointed out that Barker did not assume any underlying fractal structure. Here work is reviewed from several workers that explicitly assumes that a reservoir possesses fractal properties.

O'Shaughnessy and Procaccia (1985), who developed expressions for diffusion on fractals, advanced the basis for this work. Chang and Yortsos (1990) then advanced this work by developing expressions for flow in fractured media based on an underlying fractal structure of the aquifer. A brief description of their approach is given below, building the basis for exploring the properties of fractal media in later parts of this study.

2.5.2 Theory of flow equations in fractal media

The foundation: Fractal scaling of mass density, porosity and hydraulic conductivity

It was shown before that one of the key properties of fractal structures is that parameters such as mass, volume or density etc. decrease as described by a power law when an increasingly larger region is measured. This variation of density with respect to distance is illustrated for the 2-D case in figure 19 and is similar to Barker's (1988) varying cross-sectional flow area.

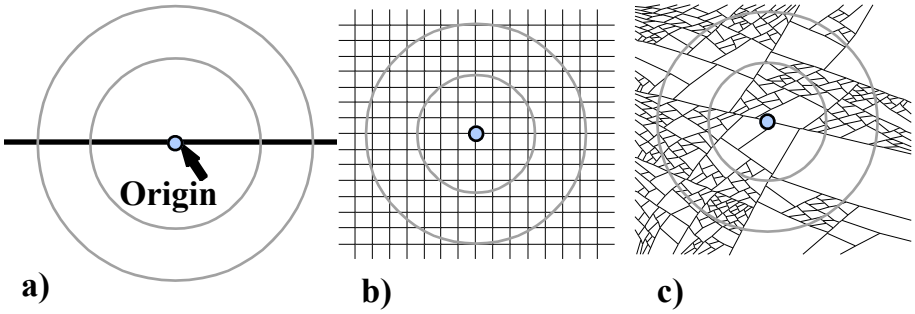


Figure 19: This figure illustrates mass scaling on different networks with 2-D embedding dimension. a) is shows a single fracture where the fractal mass dimension is one (Euclidean), b) shows a homogeneous grid of mass dimension two (also Euclidean) and c) shows a (statistically self-similar) fractal medium with a mass scaling between one and two (fractal). The fractal network in c) was created using an iterated function systems approach as described in section 3.1.

Figure 19a illustrates a single fracture. Selecting an arbitrary origin and taking the mass or volume M of the fracture contained within the radius r and area A, it is clear that M r and A r^2, resulting in a "density" of (r) = M/A with r^{-1}.

Obviously, for a fractal with mass fractal dimension D embedded in an Euclidean embedding dimension E, the density is scaled as

$$(r) = r^{D-E} \quad \text{.....(22)}$$

because M r^D and the volume V r^E.

This can be compared with the homogeneous network shown in figure 19b, where D=E=2. This results in a constant mass density. The fractal network shown in figure 19c shows the intermediate, fractal case, where 1<D<2 and E=2. If the structure shown in figure 19c has structure on every scale and is of infinite extent, above relationships will hold for any arbitrary

origin. Finite size limitations as discussed in section 2.3.6 will result in the density becoming a constant for above the upper cut-off length and below the lower cut-off length.

Assigning the point value of porosity to be zero outside the fracture (matrix) and one inside the fracture, it follows directly from (22) that

$$\phi(r) = \phi_0 \left(\frac{r}{r_0} \right)^{D-E} \quad \text{.....(23)}$$

where $\phi(r)$ is the domain porosity of a region of size r and ϕ_0 the value at $r=r_0$ and r_0 is the lower cut-off scale (the smallest fracture block size above which the object is fractal). Note the difference to the Euclidean case, where porosity is constant.

In addition to ϕ, which is related to storativity, effective hydraulic conductivity also scales with distance r:

$$K(r) = K_0 \left(\frac{r}{r_0} \right)^{D-E-\theta} \quad \text{.....(24)}$$

as described by Chang and Yortsos (1990).

Here θ is a transport exponent parameter (Acuna and Yortsos, 1995a) that does not appear in the approach used by Barker (1988). It is related to diffusion slowdown on fractal networks. See section 2.5.3 for a full discussion of this parameter.

Fractal flow equations

Using the scaling relationships given above and extending work done by O'Shaughnessy and Procaccia (1985), Chang and Yortsos (1990) derived the following equation for the response of a well producing at a constant rate using a generalised diffusivity equation:

$$p_D\left(r_D, t_D\right) = \frac{r_D^{(2+\theta)(1-\theta)}}{(2+\theta)\Gamma(\theta)} \Gamma\left(1, \frac{r_D^{2+\theta}}{(2+\theta)^2 t_D}\right) \quad \text{..... (25)}^{3}$$

---

[3] Note that in the papers by Acuna and Yortsos (1995a, 1995b) this equation is stated as $p_D\left(r_D, t_D\right) = \frac{r_D^{(2+\theta)(1-\theta)}}{(2+\theta)\Gamma(\theta)} \Gamma\left(-1, \frac{r_D^{2+\theta}}{(2+\theta)^2 t_D}\right)$, i.e. the sign of the first argument of the incomplete gamma function is reversed. Note that $\Gamma(-a,x) \neq \Gamma(a,z)$ and comparing a plot of their function definition with published plots, the equation seems to be wrong and it should be written as given by (25).

where p_D is the dimensionless pressure drop (drawdown) at dimensionless distance r_D and time $t_{D,}$ and

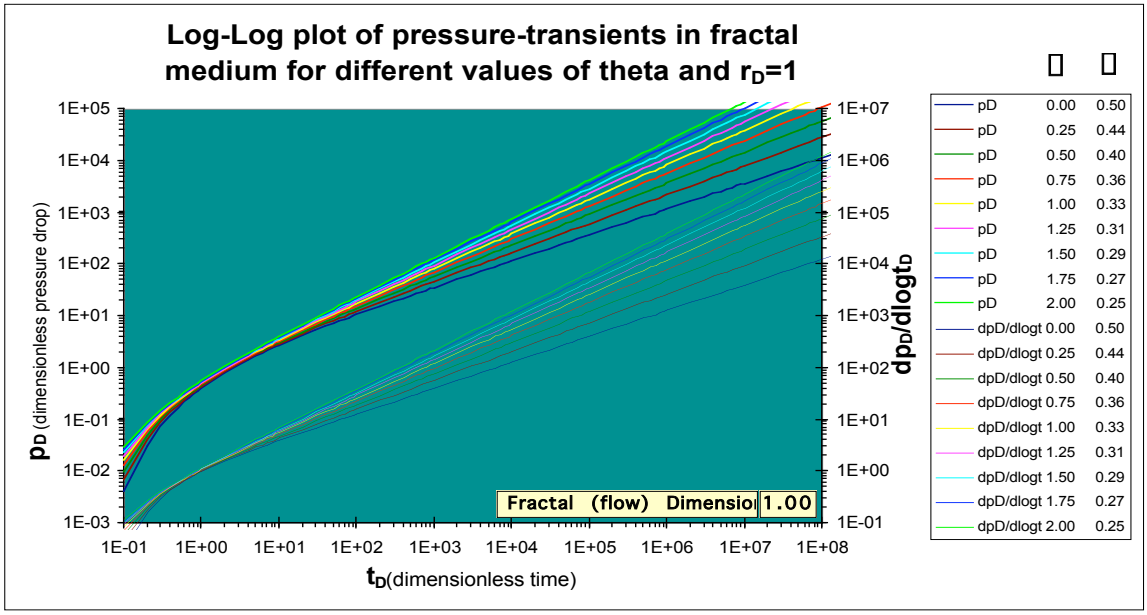
$$\delta = \frac{d_{mf}}{2+\theta} \quad \text{.....(26)}$$

with d_{mf} = mass fractal dimension.

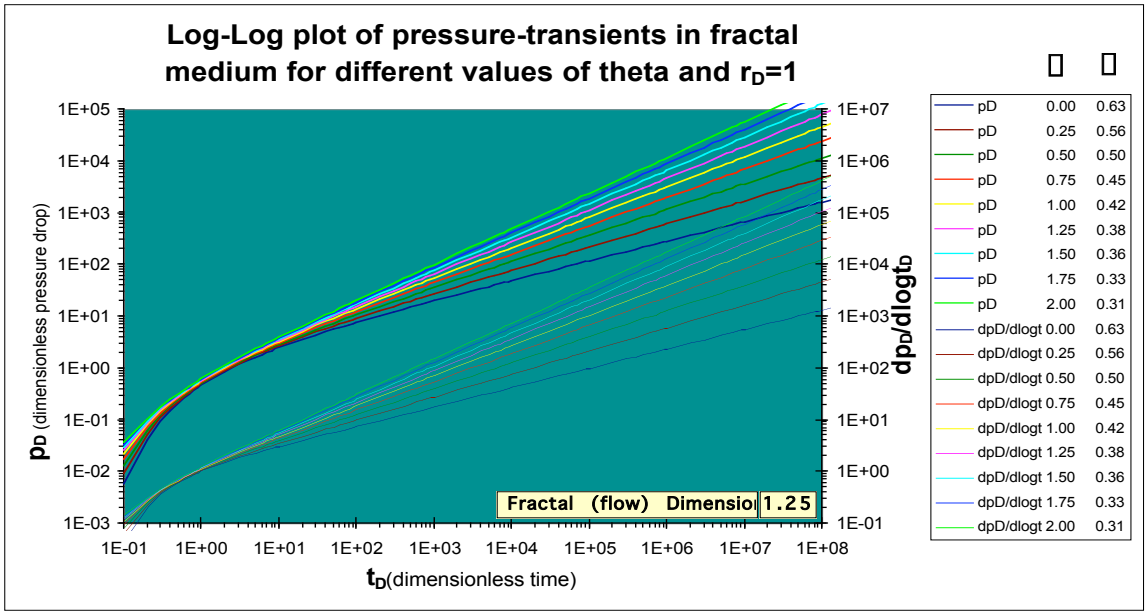
Note that also $\delta = \frac{d_s}{2}$ with d_s=spectral dimension (Acuna and Yortsos, 1995a).

It is now very instructive to compare equation (25) with the formulation of Barker as described by equation (10). Setting θ=0 and identifying γ with 1-δ it can be seen that the two equations, even though they are derived using two completely different approaches, are exactly equivalent. Hence, equation (25) describes the same variable flow dimension as Barker's but also takes into account diffusion slowdown as expressed by the parameter θ. The effects of this parameter are shown in figure 20. Note that different combinations of theta and d_{mf} can lead to the same effective value of delta.

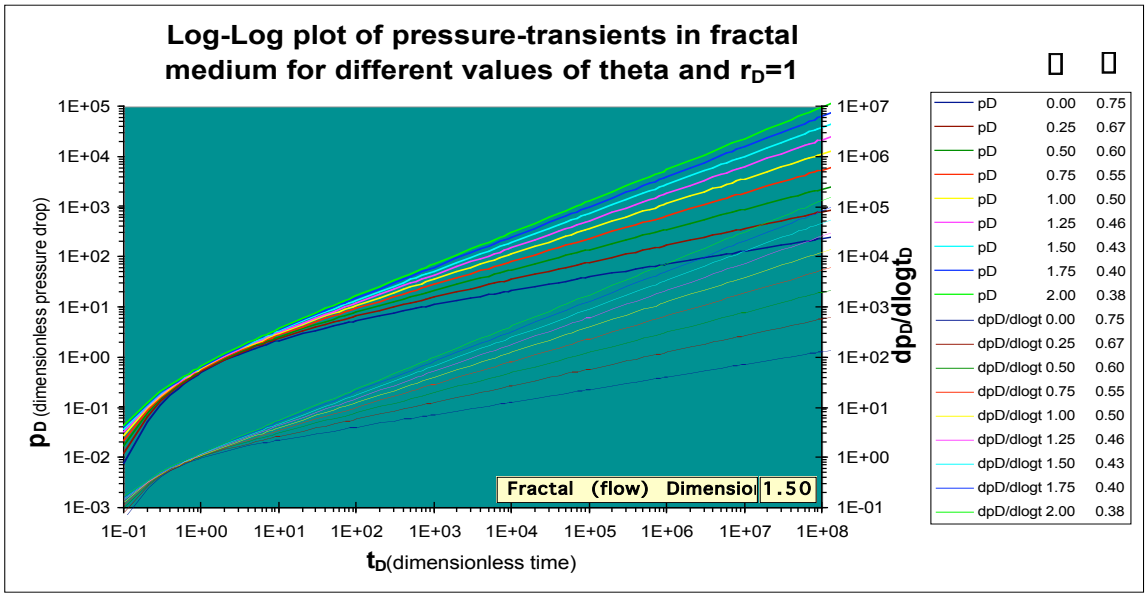
a)



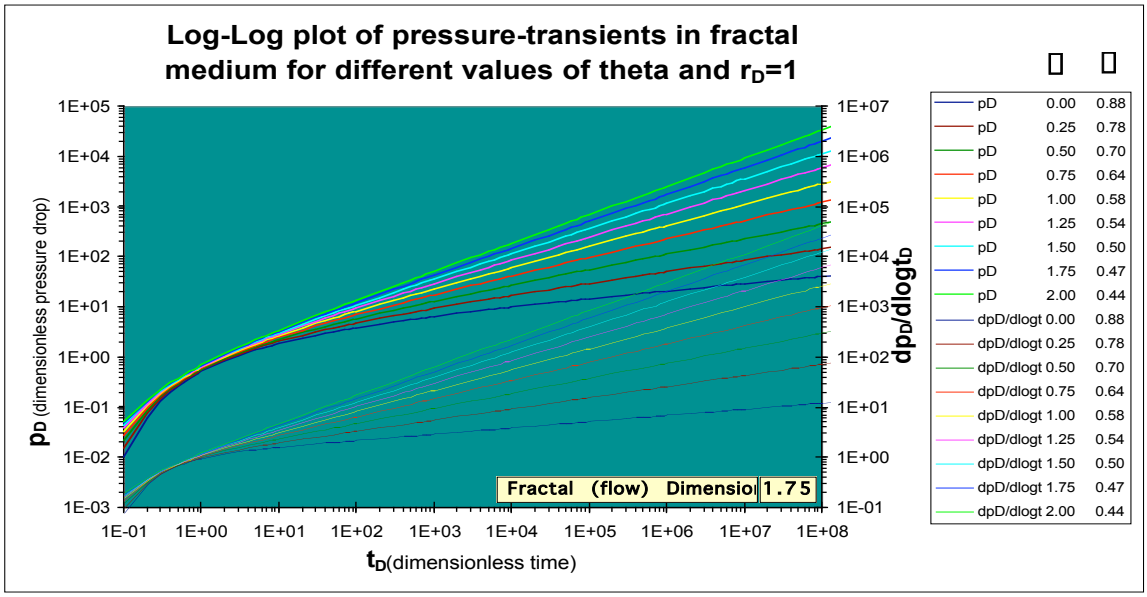
b)


Log-Log plot of pressure-transients in fractal medium for different values of theta and r_D=1
p_D (dimensionless pressure drop)
$dp_D/dlogt_D$
t_D(dimensionless time)
Fractal (flow) Dimensio 1.25
pD 0.00 0.63
pD 0.25 0.56
pD 0.50 0.50
pD 0.75 0.45
pD 1.00 0.42
pD 1.25 0.38
pD 1.50 0.36
pD 1.75 0.33
pD 2.00 0.31
dpD/dlogt 0.00 0.63
dpD/dlogt 0.25 0.56
dpD/dlogt 0.50 0.50
dpD/dlogt 0.75 0.45
dpD/dlogt 1.00 0.42
dpD/dlogt 1.25 0.38
dpD/dlogt 1.50 0.36
dpD/dlogt 1.75 0.33
dpD/dlogt 2.00 0.31

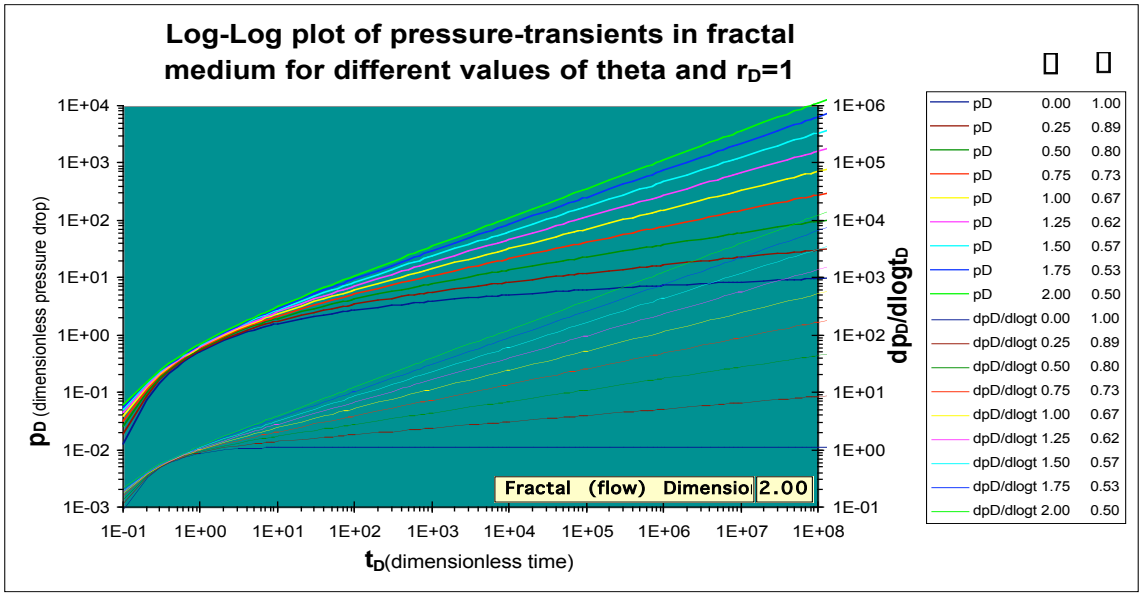
c)


Log-Log plot of pressure-transients in fractal medium for different values of theta and r_D=1
p_D (dimensionless pressure drop)
$dp_D/dlogt_D$
t_D(dimensionless time)
Fractal (flow) Dimensio 1.50
pD 0.00 0.75
pD 0.25 0.67
pD 0.50 0.60
pD 0.75 0.55
pD 1.00 0.50
pD 1.25 0.46
pD 1.50 0.43
pD 1.75 0.40
pD 2.00 0.38
dpD/dlogt 0.00 0.75
dpD/dlogt 0.25 0.67
dpD/dlogt 0.50 0.60
dpD/dlogt 0.75 0.55
dpD/dlogt 1.00 0.50
dpD/dlogt 1.25 0.46
dpD/dlogt 1.50 0.43
dpD/dlogt 1.75 0.40
dpD/dlogt 2.00 0.38

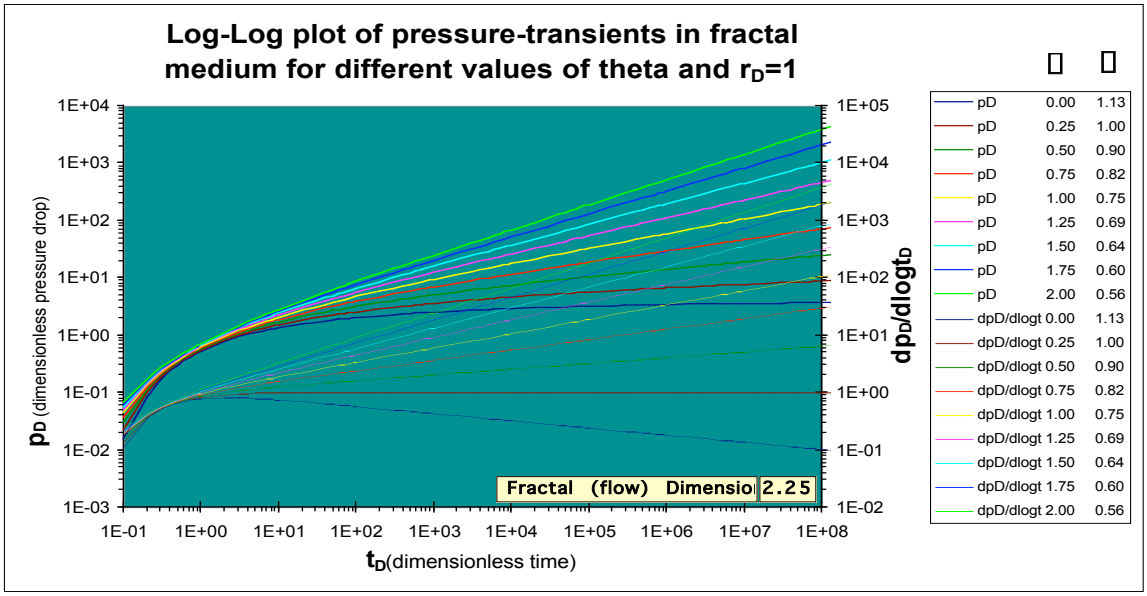
d)


Log-Log plot of pressure-transients in fractal medium for different values of theta and r_D=1
p_D (dimensionless pressure drop)
$dp_D/dlogt_D$
t_D(dimensionless time)
Fractal (flow) Dimensio 1.75
pD 0.00 0.88
pD 0.25 0.78
pD 0.50 0.70
pD 0.75 0.64
pD 1.00 0.58
pD 1.25 0.54
pD 1.50 0.50
pD 1.75 0.47
pD 2.00 0.44
dpD/dlogt 0.00 0.88
dpD/dlogt 0.25 0.78
dpD/dlogt 0.50 0.70
dpD/dlogt 0.75 0.64
dpD/dlogt 1.00 0.58
dpD/dlogt 1.25 0.54
dpD/dlogt 1.50 0.50
dpD/dlogt 1.75 0.47
dpD/dlogt 2.00 0.44

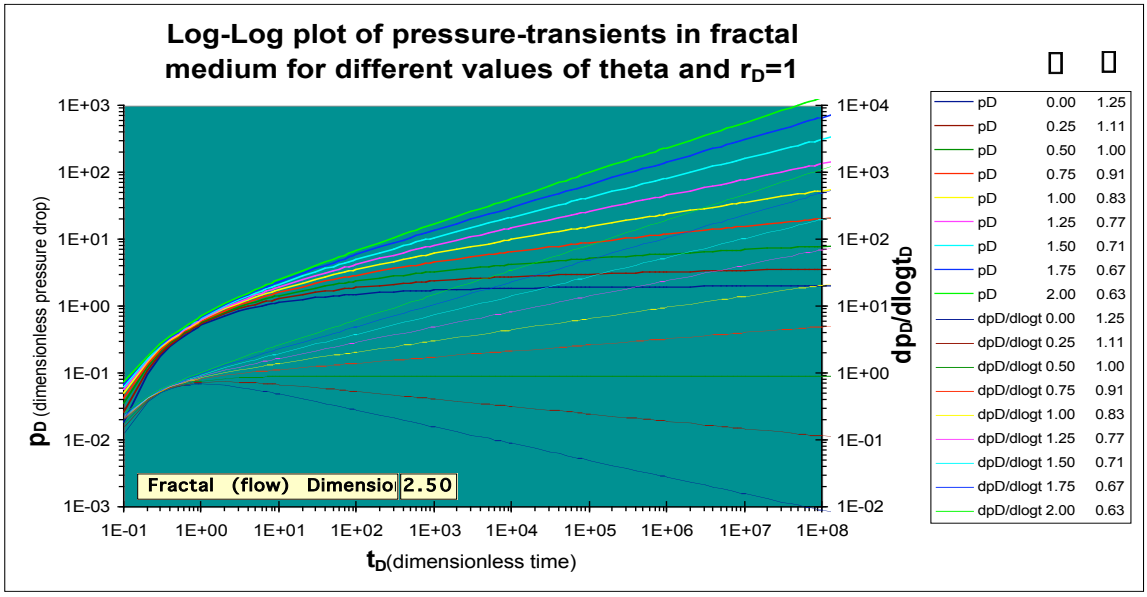
e)


Log-Log plot of pressure-transients in fractal medium for different values of theta and r_D=1
p_D (dimensionless pressure drop)
t_D(dimensionless time)
$dp_D/dlogt_D$
Fractal (flow) Dimensio 2.00
pD 0.00 1.00
pD 0.25 0.89
pD 0.50 0.80
pD 0.75 0.73
pD 1.00 0.67
pD 1.25 0.62
pD 1.50 0.57
pD 1.75 0.53
pD 2.00 0.50
dpD/dlogt 0.00 1.00
dpD/dlogt 0.25 0.89
dpD/dlogt 0.50 0.80
dpD/dlogt 0.75 0.73
dpD/dlogt 1.00 0.67
dpD/dlogt 1.25 0.62
dpD/dlogt 1.50 0.57
dpD/dlogt 1.75 0.53
dpD/dlogt 2.00 0.50

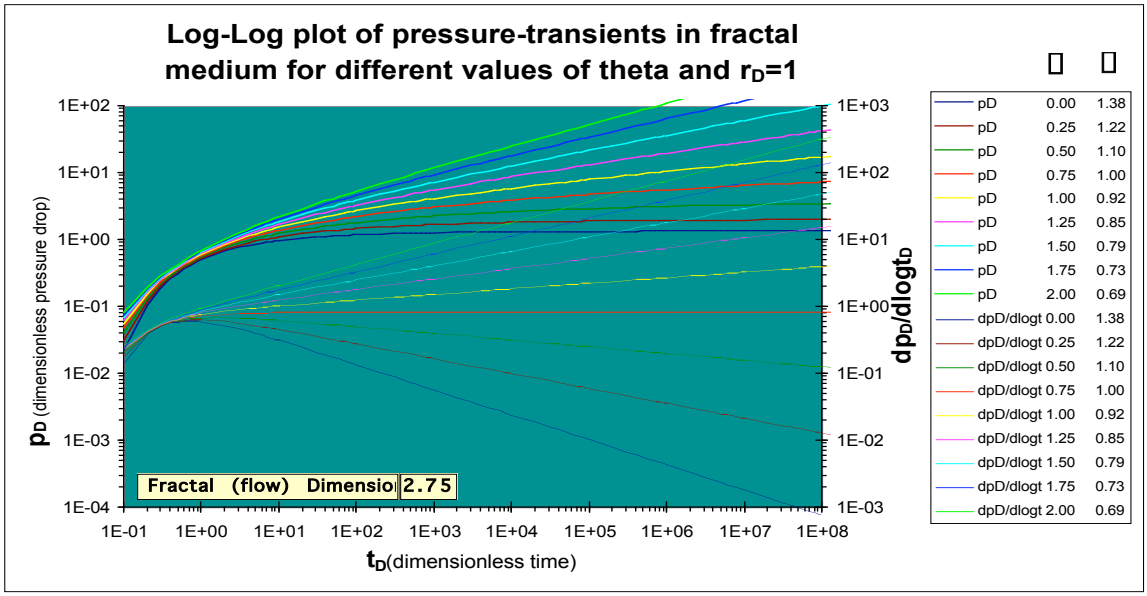
f)


Log-Log plot of pressure-transients in fractal medium for different values of theta and r_D=1
p_D (dimensionless pressure drop)
t_D(dimensionless time)
$dp_D/dlogt_D$
Fractal (flow) Dimensio 2.25
pD 0.00 1.13
pD 0.25 1.00
pD 0.50 0.90
pD 0.75 0.82
pD 1.00 0.75
pD 1.25 0.69
pD 1.50 0.64
pD 1.75 0.60
pD 2.00 0.56
dpD/dlogt 0.00 1.13
dpD/dlogt 0.25 1.00
dpD/dlogt 0.50 0.90
dpD/dlogt 0.75 0.82
dpD/dlogt 1.00 0.75
dpD/dlogt 1.25 0.69
dpD/dlogt 1.50 0.64
dpD/dlogt 1.75 0.60
dpD/dlogt 2.00 0.56

g)


Log-Log plot of pressure-transients in fractal medium for different values of theta and r_D=1
p_D (dimensionless pressure drop)
t_D(dimensionless time)
$dp_D/dlogt_D$
Fractal (flow) Dimensio 2.50
pD 0.00 1.25
pD 0.25 1.11
pD 0.50 1.00
pD 0.75 0.91
pD 1.00 0.83
pD 1.25 0.77
pD 1.50 0.71
pD 1.75 0.67
pD 2.00 0.63
dpD/dlogt 0.00 1.25
dpD/dlogt 0.25 1.11
dpD/dlogt 0.50 1.00
dpD/dlogt 0.75 0.91
dpD/dlogt 1.00 0.83
dpD/dlogt 1.25 0.77
dpD/dlogt 1.50 0.71
dpD/dlogt 1.75 0.67
dpD/dlogt 2.00 0.63

h)



i)

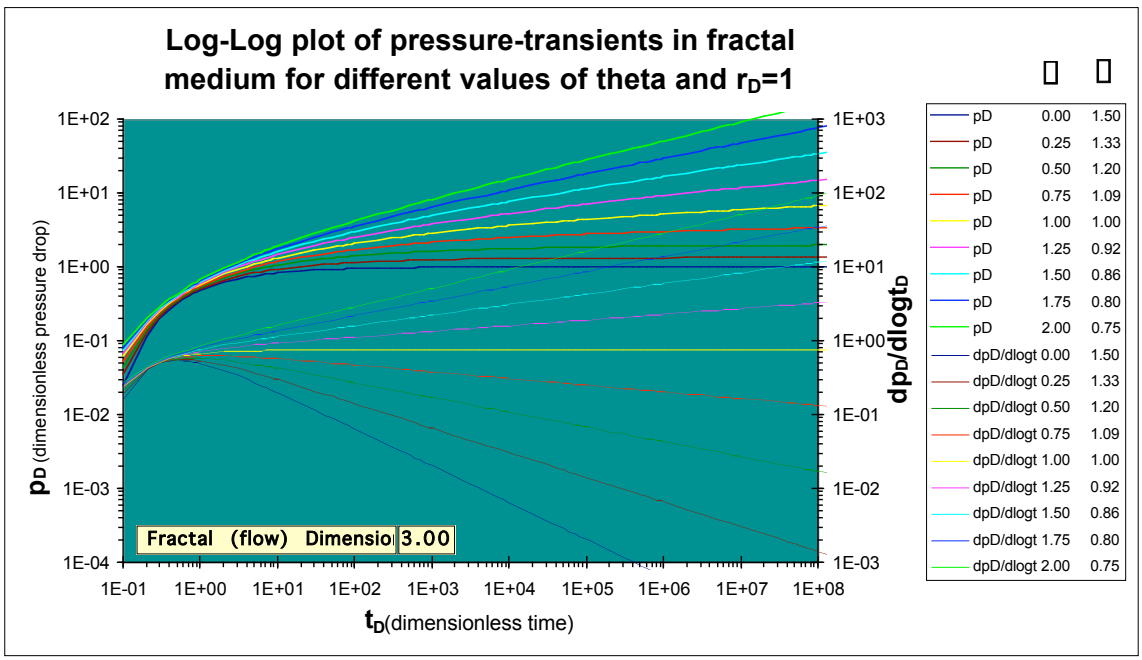


Figure 20: a) through i) illustrate the effect of the parameter on the pressure-transient response. For each flow dimension (which has been identified as fractal mass dimension), values of from 0 to 2 have been used and the resulting type curves and their derivatives w.r.t. log time are plotted in dimensionless form as in equations (3) through (5). The flow dimensions are a) 1.0 (linear flow), b) 1.25, c) 1.5, d) 1.75, e) 2.0 (radial flow), f) 2.25, g) 2.5, h) 2.75 and i) 3.0 (spherical flow). Note that for flow dimensions less than two both the pressure and the derivative curve have parallel slopes. Note also that especially for flow dimensions greater than 2.0 the effect of the parameter is such that for higher values the drawdown curve is flattened which is equivalent to a higher flow dimension, see section 2.5.3 for discussion. Note that on all graphs the pressure derivative scale is offset in order to separate the pressure curve and the derivative. All plots were created with Excel macros (see Appendix A).

Similar to the approach developed from equation 14 onwards for the approach shown by Barker it is also possible to analyse pumping curves making use of the asymptotic behaviour of equation 25. For small values of the argument $\frac{r_D^{2+\gamma}}{(2+\gamma)^2 t_D}$[4], which at the pumping well occurs after a short time, it is possible to use only the first two terms of the series expansion[5] of $\Gamma(a,x)$ to obtain

$$p_D(r_D = 1, t_D) = A + B\frac{(2+\gamma)^{1-2\gamma}}{(1-\gamma)\Gamma(\gamma)} t_D^{1-\gamma} \quad \text{.....(27)}$$

where A and B are constants.

It has been shown by Acuna and Yortsos (1995b) that the behaviour of equation 27 is fundamentally different for the two cases $\gamma<1$ and $\gamma>1$. Using equation 26 and noting that γ is always greater or equal than zero, these two cases correspond to a flow dimension less than two (intermediate between linear and radial flow) or greater than two (intermediate between radial and spherical flow). The asymptotic behaviour of 27 in both cases is described below. Note that the derivative of equation 25 can be obtained analytically as

$$\frac{dp_D}{d\log_{10} t_D} = \log_e(10)\frac{e^{-y}}{\Gamma(\gamma)(\gamma+2)^{2\gamma-1}} t_D^{1-\gamma} \quad \text{.....(28)}$$

with $y = \frac{r_D^{\gamma+2}d_{mf}}{(\gamma+2)^2 t_D}$.

- **<u>Analysis of flow behaviour for $\gamma<1$</u>**:

  It is easy to see that if $\gamma<1$, the time-independent constant A in equation 27 becomes negligible after a short time. In this case, an approximation to (27) can be written as

$$p_D(r_D = 1, t \gg) \sim Ct^{1-\gamma} \quad \text{.....(29)}$$

[4] Note that this is equivalent to the expression *u* as used by Barker.

[5] This is similar to obtaining the well known Jacob equation from the Theis series expansion.

where C is also a constant, ignoring any skin-effect. An important result is the observation that a log-log plot of this equation will show a slope of 1-θ.

Taking the derivative of 29 with respect to log t_D gives

$$\frac{dp_D}{d(\log t_D)} \sim C(1-\theta)t^{1-\theta} \quad \text{.....(30)}$$

Hence this pressure derivative will also show a slope of 1-θ on a log-log plot of dimensionless pressure vs. dimensionless time. In addition, both graphs will be offset at $\log t_D=0$ by

$$\log\left(\frac{1}{1-\theta}\right) \quad \text{..... (31)}$$

Acuna and Yortsos (1995a) report real pressure data that show this behaviour, which can also be observed from figures 6 and 20.

- **Analysis of flow behaviour for θ>1**:

When θ>1, then the mass fractal dimension is greater than two because θ is always positive (see equation 26). This is also true for Barker's flow dimension in the case of θ=0. In this case the asymptotic behaviour of equation 27 *does* depend on the constant A at large times and can be described by $p_D(r_D = 1, t \to \infty) \sim A - Ct^{1-\theta}$. Here the constant A describes the asymptotic pressure drop at large times. However, the pressure derivative is still the same as given by equation 30 and still a power-law function of time. Thus, the log-log plot of the derivative with respect to log t_D is still a straight line with *negative* slope 1-θ. This is clearly shown in figure 20 for flow dimensions greater than two. Acuna and Yortsos (1995b) proposed the following method to estimate the constant A from pumping test data: "Draw a line above and parallel to the pressure derivative curve, separated by a distance log(1/(θ-1)). At the point where this line intersects the original pressure curve, the corresponding pressure value must equal A/2 [...]".

Successful applications of these techniques on real pressure data that show fractal behaviour have now been reported by several authors, including Beier (1990) and Acuna and Yortsos (1995b).

2.5.3 Discussion of the parameter θ

It has already been mentioned that the parameter θ is related to topological network properties. Note that the fractal mass dimension (or flow dimension) in equation 25 always appears together with the parameter θ, either directly or in the form of θ, because $\theta = \frac{d_{mf}}{2+\theta}$.

Hence the proposed analysis methods described above only give θ, which cannot be directly decomposed into d_{mf} and θ. An important consequence is that a system that appears to respond in a typical homogeneous fashion might not necessarily be so. For example, given a radial homogeneous response with a pressure derivative slope of zero (θ=1), strictly the only conclusion allowed is that $d_{mf} = 2 + \theta$.

This could either be the result of a Euclidean reservoir with θ=0 and a flow dimension of two, but may equally well be caused by a fractal network where θ>0 and d_{mf}>2. Similarly, a linear well response could be caused by the interception of a single fracture or a poorly connected fracture network.

At the moment, there is no known unique relationship between d_{mf} and θ for arbitrary networks, although θ is expected to increase with higher network tortuosity and poorer connectivity (Orbach, 1986). θ also seems to be linked to the theory of percolation networks, where adjacent network elements are either open or closed to flow, according to a probability relationship. At the "*percolation threshold*", i.e. the probability where flow from one end to the other of a network is very likely, the following relationship is expected:

$\theta = \frac{d_{mf}}{2+\theta} = \frac{2}{3}$ (Alexander and Orbach, 1982; Orbach, 1986).

This conjecture infers that percolation networks at the percolation threshold result in a slope of the pumping response on a log-log plot of 1-2/3 = 1/3.

Alexander and Orbach (1982) also linked the fractal mass dimension d_{mf} and the parameter θ to the *spectral dimension* d_s of a network as defined by Mandelbrot (1977):

$$\theta = \frac{d_{mf}}{2+\theta} = \frac{d_s}{2} \quad \text{.....(32)}$$

In 3-D percolation networks it is known that θ=1.784 (Isichenko, 1992). Hence it should in theory be possible to obtain a value of theta for a given (fractal) network by simulating a self-avoiding random walk on it.

O'Shaughnessy and Procaccia (1985) showed that, given a fractal of dimension D embedded in Euclidean space of dimension E, the mean-square displacement after time t of a random walker on the fractal obeys $\langle r^2(t) \rangle \sim t^{2/(2+\theta)}$, where r is measured in the Euclidean space. In fractal networks there exist highly tortuous paths of many scales, where the random walker is effectively lost and escapes only after long times. Thus the parameter θ is related to the network connectivity and describes the deviation from a random walk in a Euclidean network due to *diffusion slowdown* in fractal structures.

Considering the Euclidean geometries in figures 19a and 19b, diffusion is Fickian, θ=0, and the characteristic distance of a random walk $\langle r^2 \rangle \quad t$ is obtained. For a fractal network, as shown in figure 19c, many traps of various scales in the network cause diffusion to slow, giving a value of $0 < \theta < 1$. A modified Sierpinski gasket, as shown in figure 16a, gives a value of $\theta \approx 0.16$.

In practice, the separate evaluation of the two fractal parameters d_{mf} and θ requires additional tests. Acuna and Yortsos (1995b) propose to analyse successive measurements of steady-state flow at different scales in addition to transient testing because for a fractal system, the steady state conductance between two points at distance L scales as $g \sim L^{-\theta-E+d_{mf}}$. Another method would be the use of numerical random walk studies on networks with fracture geometry parameters obtained from borehole data as described by Orbach (1986).

Beier (1990) developed expressions for the *early* time approximations to flow in fractal media. He asserts that at early times the slope of the pressure curve is given by $\left(1-\frac{d_s}{2}\right)\left(1-\frac{1}{d_{mf}}\right)$ where d_s is the spectral dimension. This, together with equation 29, means that the late time slope depends only on θ, while the early time slope depends on both, θ and d_{mf}. In theory,

this would allow the simultaneous determination of both, d_{mf} and θ. However, early time data are likely to be obscured by well-bore storage effects.

It is important to realise that the parameter theta is not included in the approach chosen by Barker. However, its effect is clearly important and warrants further study with the possible use as a network connectivity parameter. Software packages like FracMan, that are based on flow dimension alone, are likely to give non-unique or misleading results due to the effects of θ, especially for very sparse and/ or tortuous fracture networks as encountered in many low-permeability environments, assuming the rock is "fractal". Further work, e.g. by simultaneously fitting both parameters to real data, however, is needed to test whether the relationship above could be useful.

3. Forward modelling of fractal aquifer response

3.1 Generation of synthetic fractal fracture networks

In order to study the theoretical behaviour of (fractal) networks it is necessary to create synthetic fracture networks. Ideally, network algorithms should try to model fracturing processes that are occurring in nature as closely as possible. Percolation models are not necessarily appropriate for naturally fractured rocks because mechanisms such as shear fracturing, extension fracturing and fragmentation might occur as well, which differ from percolation.

Generally, network models should be able to incorporate available data such as fracture length and aperture distributions as obtained from borehole logs or geo-statistical descriptions. Such "universal" network models would depend on the integration of all occurring fracturing processes, which is not available at the moment. Given the scope of this study, it was considered sensible to concentrate on network algorithms that are easily implemented as computer code and allow sensitivity studies pertaining to fractal parameters as described above. In this sense, the approach described in this chapter should not be considered to closely model fractures as occurring in nature but to allow theoretical studies on fractal networks that can then be used to build more sophisticated models.

3.1.1 The iterated function system (IFS) approach

Turcotte (1986) describes a fragmentation process that leads to fractal block size distributions. A simple algorithm that creates fragments leading to a power-law distribution is described below. Heavily modified and debugged algorithms and code supplied by Jorge Acuna (Unocal) were used and his kind help is acknowledged here. First, the code is used to generate *exact* fractal fracture networks such as a modified Sierpinski gasket and this is later extended to stochastic (statistically self-similar) fractals. Both network types can be analysed geometrically and the results can then be compared to the flow dimension obtained from hydraulic modelling.

Barnsley (1988) first introduced “iterated function systems” as a method to build finite two-dimensional networks by the successive application of simple rules of division. A fractal is obtained from an initial shape (the *initiator*) by iteratively applying a set of numerical transforms (the *propagator*). The original shape is mapped onto consecutively smaller shapes that are added to the previous image and contribute to the overall structure. Combining this algorithm with rules that select only parts of an image for further mapping lead to the approximation of a fractal structure after several iterations. This is illustrated in figure 21.
To create self-similar patterns, two numerical transformations are applied: a rotation and a subsequent reduction of the initial figure. Each new generation has twice the number of fracture segments as the previous one. The transformation is achieved in 2-D by the bi-linear equations

$$x_n = a + bx_{n-1} + cy_{n-1} + dx_{n-1}y_{n-1} \quad \text{.....(33)}$$

$$y_n = e + fx_{n-1} + gy_{n-1} + hx_{n-1}y_{n-1}$$

where the index n denotes the current fracture generation, n-1 the previous generation, x and y are co-ordinates of segment end-points and a through h are constant coefficients that are determined by the geometry. Note that this mapping can be very easily extended to 3-D by introducing four more coefficients. In 2-D, the four corners of each quadrilateral provide the parameters necessary to obtain the coefficients, i.e. the original square of figure 21 is mapped onto the quadrilateral given by the first fracture and the upper two corners of the square etc. When d=h=0, the transformation is linear and self-affine. If, in addition, b=g and c=-f, the shape is self-similar (Acuna and Yortsos, 1991). This method allows great flexibility in defining geometrically complex networks. In the simplest case, this method can be used to create regular 2-D networks by using a first initial fracture that is parallel to either axis of the original square. A power-law relationship of the block-size can be introduced by using algorithms that determine whether any given block is allowed to fracture further or not. Numerically, the network is represented by individual node co-ordinates which define the start and end points of line segments (bonds). In this study, the iterated function system method was applied to generate and study two different kinds of networks which are described next.

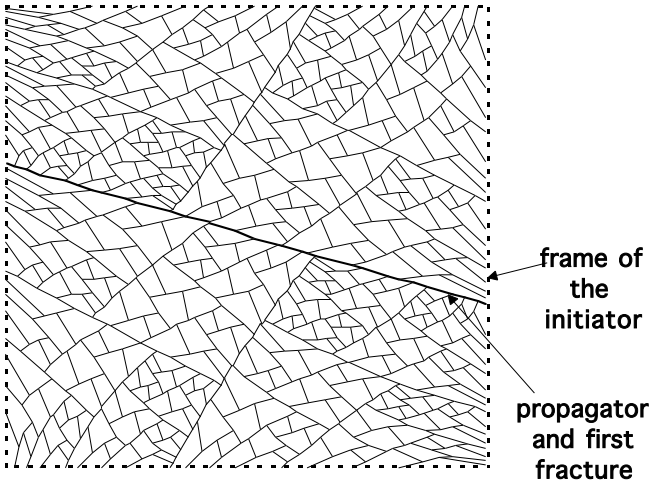


Figure 21: This figure illustrates the iterative generation of a self-similar network. The original square frame is transformed into successively smaller quadrilaterals that are determined by the angle of the first planar fracture. Here, eight iterations were performed. Note that higher generation fractures are warped by the transformation.

3.1.2 A modified Sierpinski gasket

In this study the first starting point in the analysis of fractal networks was the generation of an exact network pattern. A fractal pattern known as the "modified Sierpinski gasket", as introduced by Sammis (1987), was chosen because it has known fractal properties, is relatively easy to implement and is amenable to analytical studies. The modified Sierpinski gasket in 2-D and 3-D has already been used as an example in figure 16a. It is constructed in 2-D by taking a square as the initial shape and then subdividing it into four smaller squares. Of these four squares, three squares are subdivided again and the same process is then repeated. The fractal mass dimension of this shape is given by $\frac{\ln 3}{\ln 2} \approx 1.58$. Using the IFS pattern generator, this pattern can be generated by not further subdividing appropriate fractures for each second generation of new fractures and connecting the outside fractures with a "border frame". Figure 22 shows a simple modified Sierpinski gasket with 3 fracture generations and 2 different block size generations.

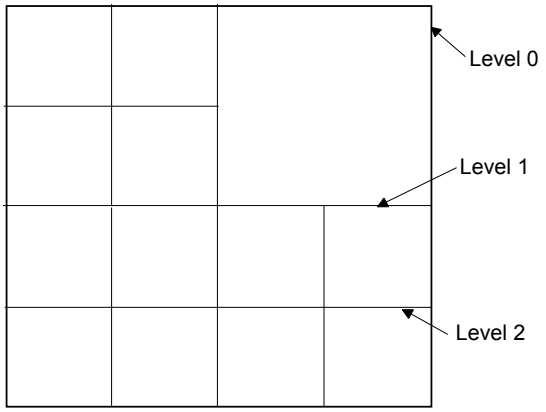


Figure 22: A modified Sierpinski gasket with just 3 fracture generations and 2 block generations. Different generations are annotated.

3.1.3 Flow area properties of recursive networks

One advantage of using *regular* fractal structures is the possibility of deriving basic geometric properties and comparing these to the resulting flow in such networks. Barker (personal communication) developed simple expressions that relate the fractal structure to parameters such as number of bonds and flow area as a function of radius. His approach has been used below.

The required parameters to calculate bond densities etc. are:

a = number of bonds added to a unit cell when adding a level

b = number of filled cells at each level

c = ratio of bond lengths at levels n and n+1 (>1)

M = number of levels within the outer boundary of the network

l = length of smallest bond

The total number of bonds inside the bounding shape can be calculating by realising that by adding one level of subdivision, existing bonds are multiplied according to the ratio of bond-lengths, but only in those cells that are subdivided further. This results in the expression for the total number of internal bonds as $N_b = a\sum_{k=1}^{M} c^{M-k} b^{k-1} = a\left[\frac{b^M - c^M}{b - c}\right]$.

For example, the Sierpinski gasket of figure 22 has a=4, b=3, c=2 and M = 2. Hence the number of internal bonds is 20.

One outer edge of a network will have the length $L = c^M l$ which leads to the general area $A = \alpha(c^M l)^2$, where α is a geometry factor, e.g. 1 for a square network. Hence, the average bond density within the network is $\rho = N/A = \frac{a}{\alpha l^2 c^{M+1}} \left[\frac{(b/c)^m - 1}{b/c - 1} \right]$.

Barker (personal communication) extended this to obtain the bond density as a function of radius. The bond density at a given radius can then be converted to an effective flow area of a homogeneous medium. The resulting flow dimension is given by $d = \frac{\ln b}{\ln c}$, which is, comparing with equation 21, equivalent to the fractal mass dimension of the network.
If the scaling of the flow area with respect to radius is the main parameter characterising flow then, according to equation 15, the pumping response of a well situated at the centre of a Sierpinski network should show a slope of $1 - \frac{\ln 3/\ln 2}{2} = 1 - \frac{1.58}{2} = 0.21$. However, this is only true if the parameter theta is zero. This can be tested by simulating a pumping test using a generated Sierpinski network and this is described in the next section.

3.2 Simulating a pumping test on a regular fractal network

A pumping test simulation was conducted on a 10m by 10m Sierpinski grid with a smallest blocksize of $\frac{10m}{2^7} = 0.0781m$. The technical details of the implementation of the IFS grid generator, adaptation of the grid output for input into a finite element code and visualisation tools are partly discussed in appendix B and in the code supplied in electronic form.
The fracture apertures were set uniformly to 4.964706×10^{-05}m for each fracture segment, which is equivalent to a fracture transmissivity of $T=1.0 \times 10^{-7} m^2/s$ using the cubic law as shown in equation (36). Since there is no fractal scaling of the aperture width, according to Barker's formulation the flow dimension should only depend on the blocksize distribution and its fractal dimension. Fracture storage was set to $S_s=5.0 \times 10^{-2}$ to extend the drawdown time. Since the finite element code used requires fixed head conditions around the boundary, this also serves the purpose to delay the effects of the constant head boundary.

Discussion of results

The results of the pumping test simulation are illustrated in figures 18 through 28. These shows the network, pressure transients for the pumping well and four observation wells together with the pressure derivative w.r.t. logarithmic time. Also plotted is the slope of the pressure transient in order to determine flow dimension and the value of θ. The slope is obtained by linear fitting a straight line to a central point and two neighbours on either side. The finite element code used requires that the outer network boundary is set to a constant head. This results in three different regions of flow on the graphs: an initial period of flow stabilisation (I), a straight line (fractal) response (II) and a region where the effects of the constant head boundary are seen (III). The latter is particularly well exhibited by the derivative plot as it tails of to zero at late time.

At the pumping well, the slope of the pressure transient curve as well as the slope of the derivative w.r.t. log time are both approximately 0.27. Using Barker's formulation, the application of equation 12 results in a flow dimension of $n = 2(1-\theta) = 2(1-0.27) = 1.46$. This is markedly different from the theoretical value of 1.58 for the flow area scaling of a Sierpinski gasket as shown in section 3.1.3.

Instead, by *assuming* that the fractal mass dimension is d_{mf}=1.58, the value of theta for this network can be evaluated from $1-\theta = 0.27$ and by using equation (26). This gives a value of $\theta \approx 0.16$. A very similar value has been reported by Acuna and Yortsos (1995a), who simulated a random walk on this type of network as described in section 2.5.3 which confirms this result. Additionally, the value of the spectral dimension can be calculated as $d_s = 2\theta = 2(1-0.27) = 1.42$, using equation 32. Interestingly, this value is similar within error to the value of 4/3 reported by Orbach (1986) for networks at the percolation threshold. This indicates that the modified Sierpinski gasket that was used as fracture network behaves similar to a percolation network just above the percolation threshold. Intuitively, this is related to the very sparse connection of the pumping well in figure 23 towards the bottom left half of the network.

A significant result is that the parameter θ is important to characterise the flow in fractal networks. In a real pumping test, the geometry of the fracture will not generally be known a priori and hence it is only possible to estimate a value for θ and not the flow dimension.

Observation wells close to the pumping well show a similar response as described above, although the small number of fracture generations used in this model, and the presence of boundary effects, influence the pressure response and only a short period of a "fractal" straight line behaviour is observed. Although this is purely the result of the limitations of the model used, it becomes clear that real pumping tests designed to elucidate flow dimension or fractal behaviour might have to be conducted for a considerably longer period than usual for tests that mainly aim to give values for transmissivity.

Note also, that the pressure derivative curves of the observation wells show a different offset compared with the pumping well. Although in theory the derivative curve should be offset with respect to the pressure transient curve by the amount 1-□ (see equation 31), figure 25 shows that the derivative is offset upwards rather than downwards. Two interpretations of this result are possible: Due to the short time where truly "fractal" flow takes place, boundary and finite size effects of the model introduce this behaviour as artefact. Alternatively, this effect is linked to the way the model is represented as a discrete grid. In theory the drawdown curve can be described by an equation of the form $h_D \quad Ct_D^{1\square\square}$ (see equation 27), where h_D is dimensionless drawdown, t_D dimensionless time and C a constant, and hence the derivative w.r.t. log time should be of the form $\frac{dh_D}{d\log t} \quad C(1\square\square)t_D^{1\square\square}$, which leads to straight lines of slope $(1\square\square)$ and offset $\log(1\square\square)$ on a log-log plot. However, the pumping well of the model, represented here as a node, is not an ideal line sink and it is difficult to assign a radius to the well on the grid. Instead, the effects of the immediate neighbour-nodes will lead to a behaviour that is better described by an equation of the form $h_D \quad Ct_D^{1\square\square} + A$, where A might vary with time. On differentiating and plotting on a log-log plot, this will lead to a different offset of the drawdown derivative with respect to log time.

Generally, it can be seen from the different observation wells, that the identification of a fractal flow regime becomes more difficult further away from the pumping well in the presence of a boundary. It would be interesting to see results from a constant flux boundary, however, there was not enough time in this study to attempt the necessary modifications.

From the results presented here, it becomes clear that fractal flow behaviour *can* be observed at the pumping well if the underlying fracture network has fractal geometric properties. This will be further investigated in the section covering inverse modelling.

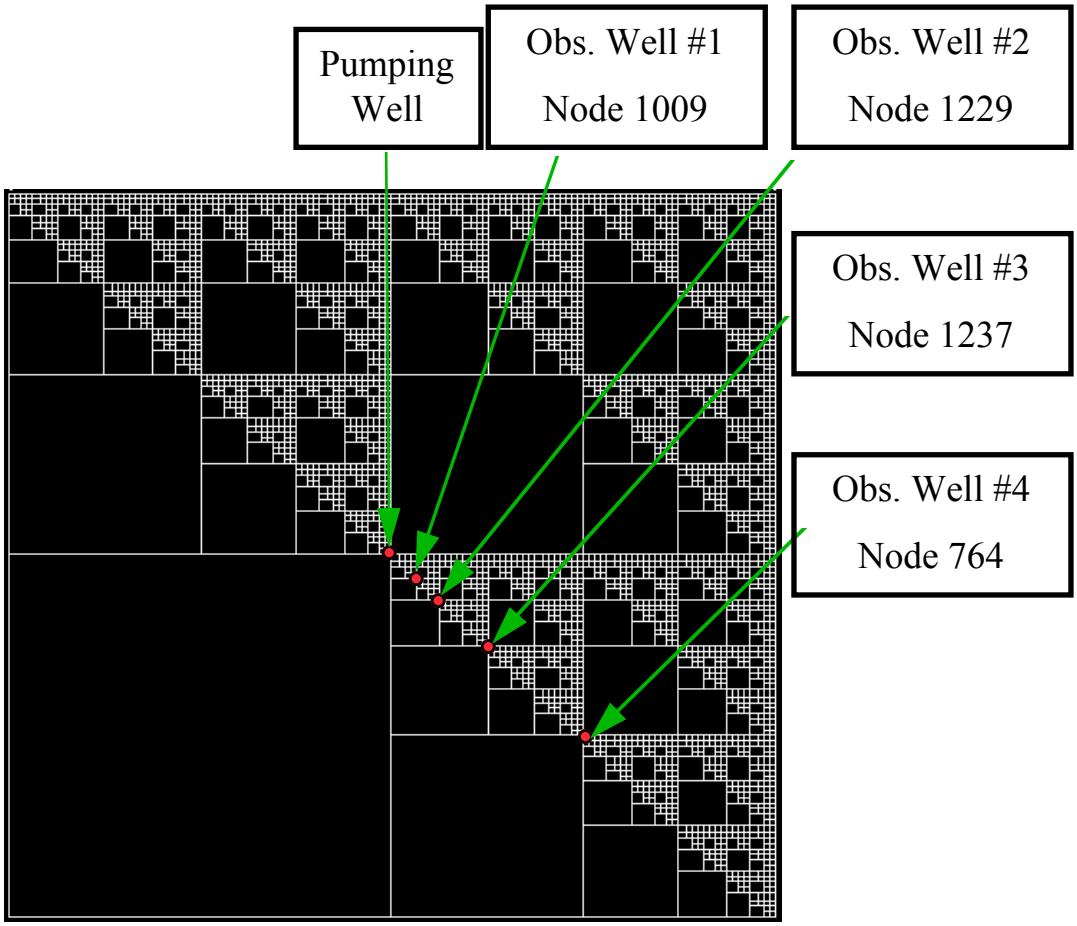


Figure 23: The modified Sierpinski gasket used to simulate flow on a fractal network. The network has 13 fracture generations and 7 different block sizes. Shown are the position of the pumping well and four observations wells, identified by their node number. The grid size is 10m by 10m.

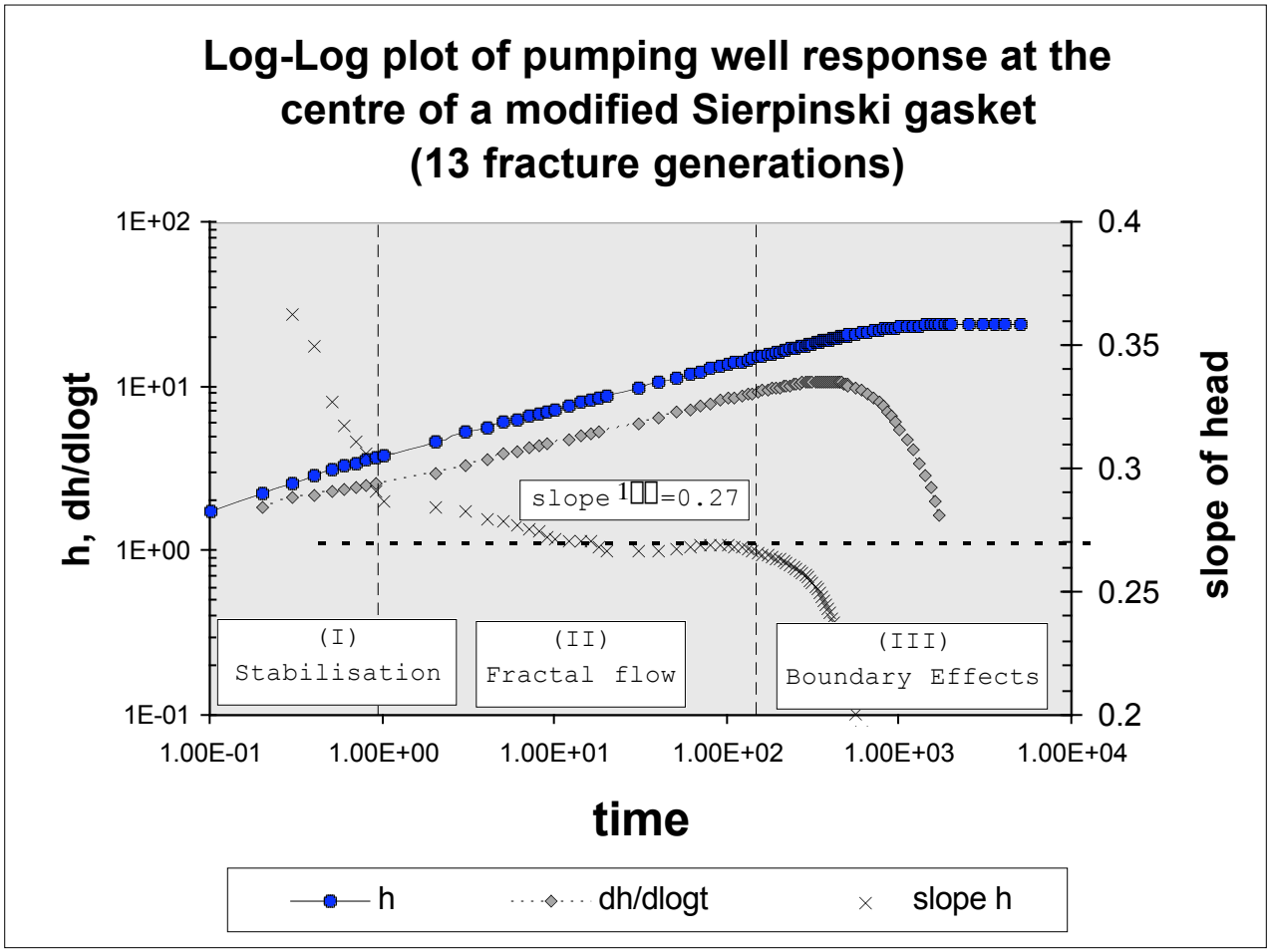


Figure 24: Shown is a log-log plot of drawdown vs. time at the pumping well. Note that, after a short stabilisation period, where individual fractures determine the flow, the drawdown curve as well as the drawdown derivative with respect to log time show the behaviour predicted for flow in a fractal medium: Both, drawdown and derivative plot on parallel and straight lines. At late times, the effect of the constant head boundary is seen where the drawdown and the derivative tail off. The slope of the straight part was determined by a plot over five points of a regressional best fit. The value of this slope can be approximated as ~0.27.

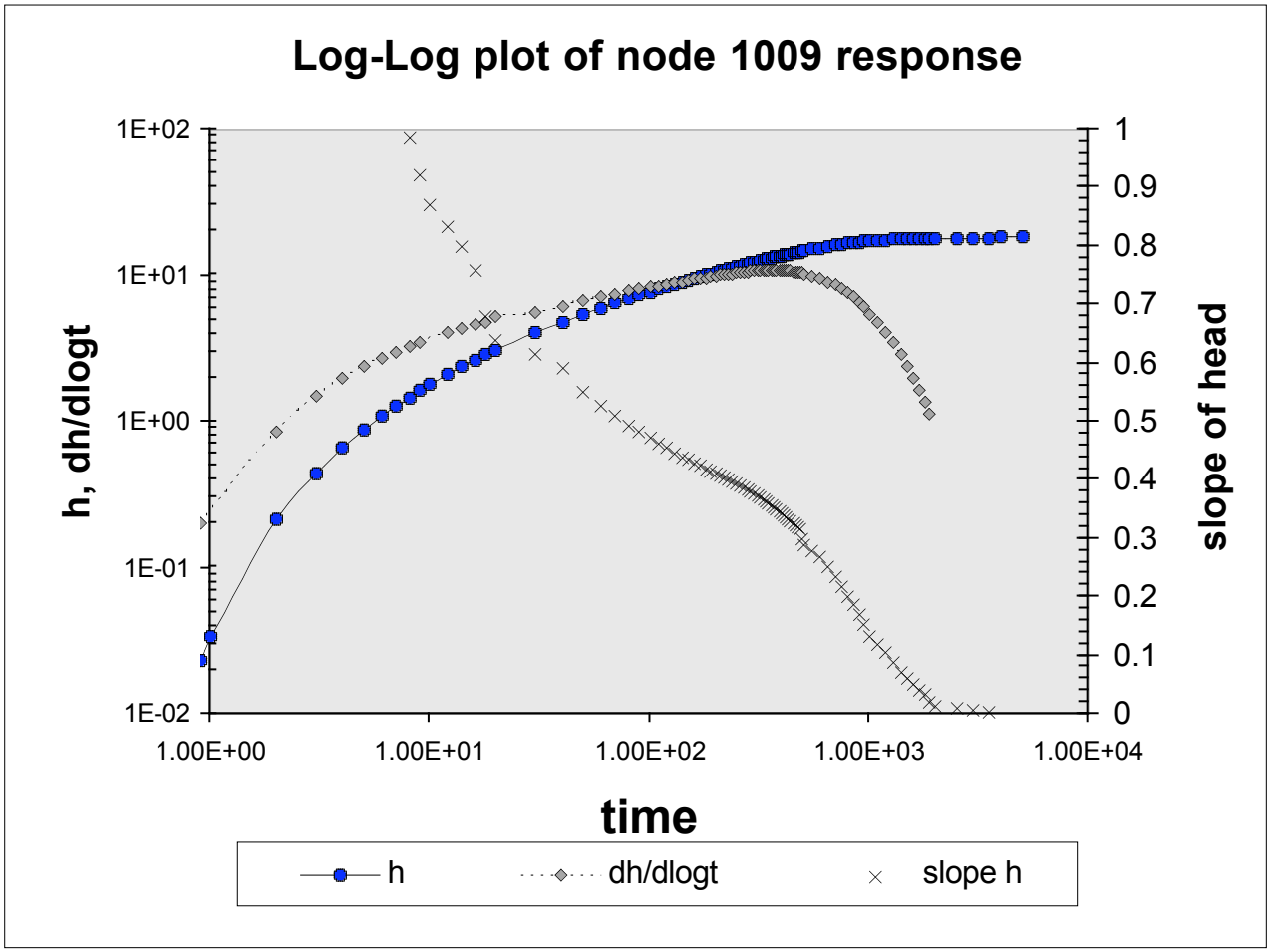


Figure 25: Log-log plot of drawdown and drawdown derivative w.r.t. log time versus time for the observation well closest to the pumping well. Note that the response deviates from the theoretical response. The drawdown curve still shows a straight-line section and so does the derivative. However, they are not parallel anymore. Note also that the drawdown derivative curve now plots *above* the drawdown curve for the largest part of the plot, contrary to theory. This has been explained in the discussion of results above and might be the result of discretisation of the model on a network.

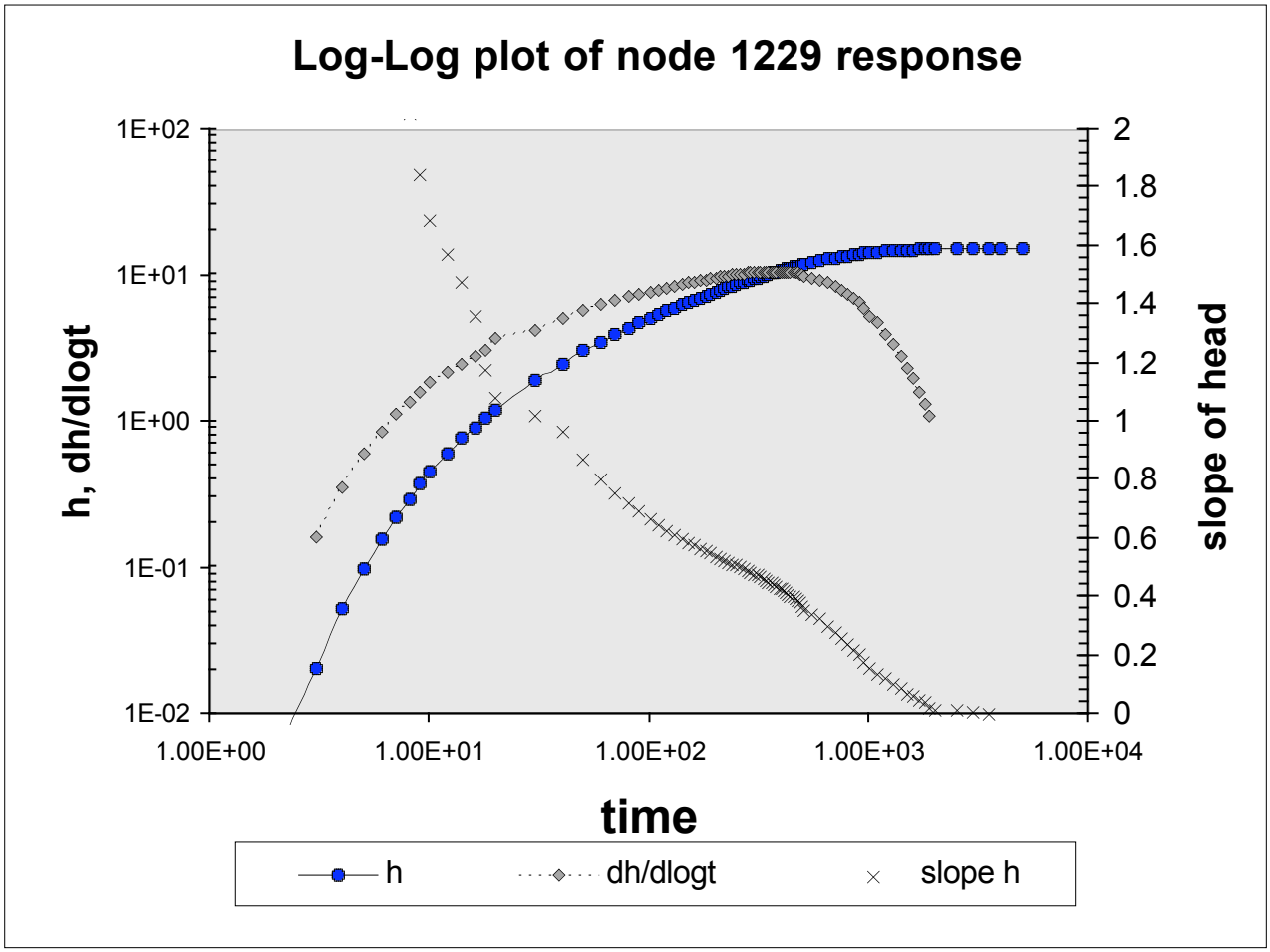


Figure 26: Log-log plot of drawdown and drawdown derivative w.r.t. log time versus time for the second closest observation well. It can be seen that the drawdown curve exhibits a straight-line section at intermediate time, but the derivative curve deviates from this theoretical response. The effects of the boundary can be seen to affect the drawdown more strongly than before.

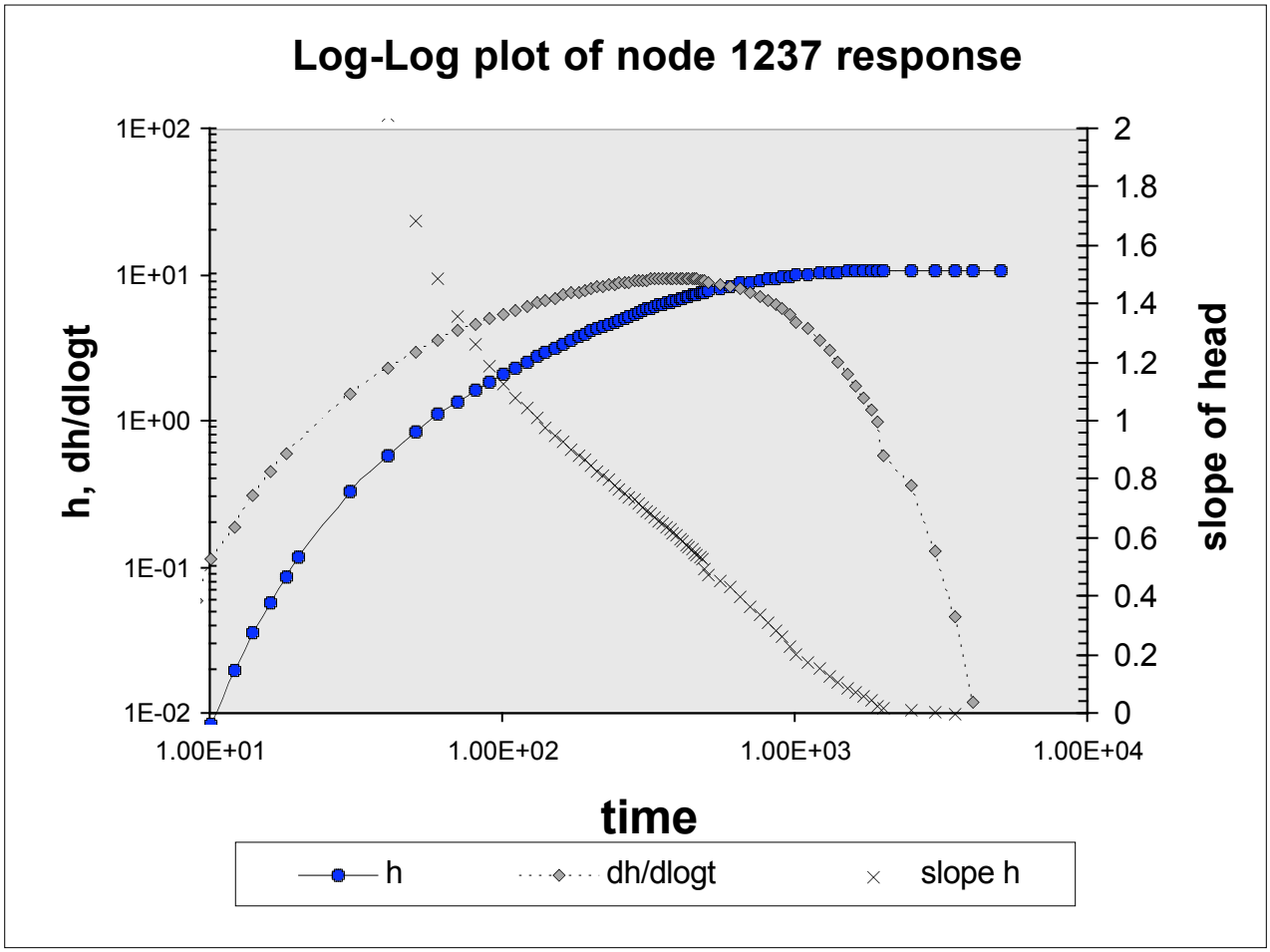


Figure 27: Log-log plot of drawdown and drawdown derivative w.r.t. log time versus time for the third closest observation well. The straight-line section of the drawdown curve is now shorter than before and the effects of the boundary can be seen to affect the drawdown more strongly than before.

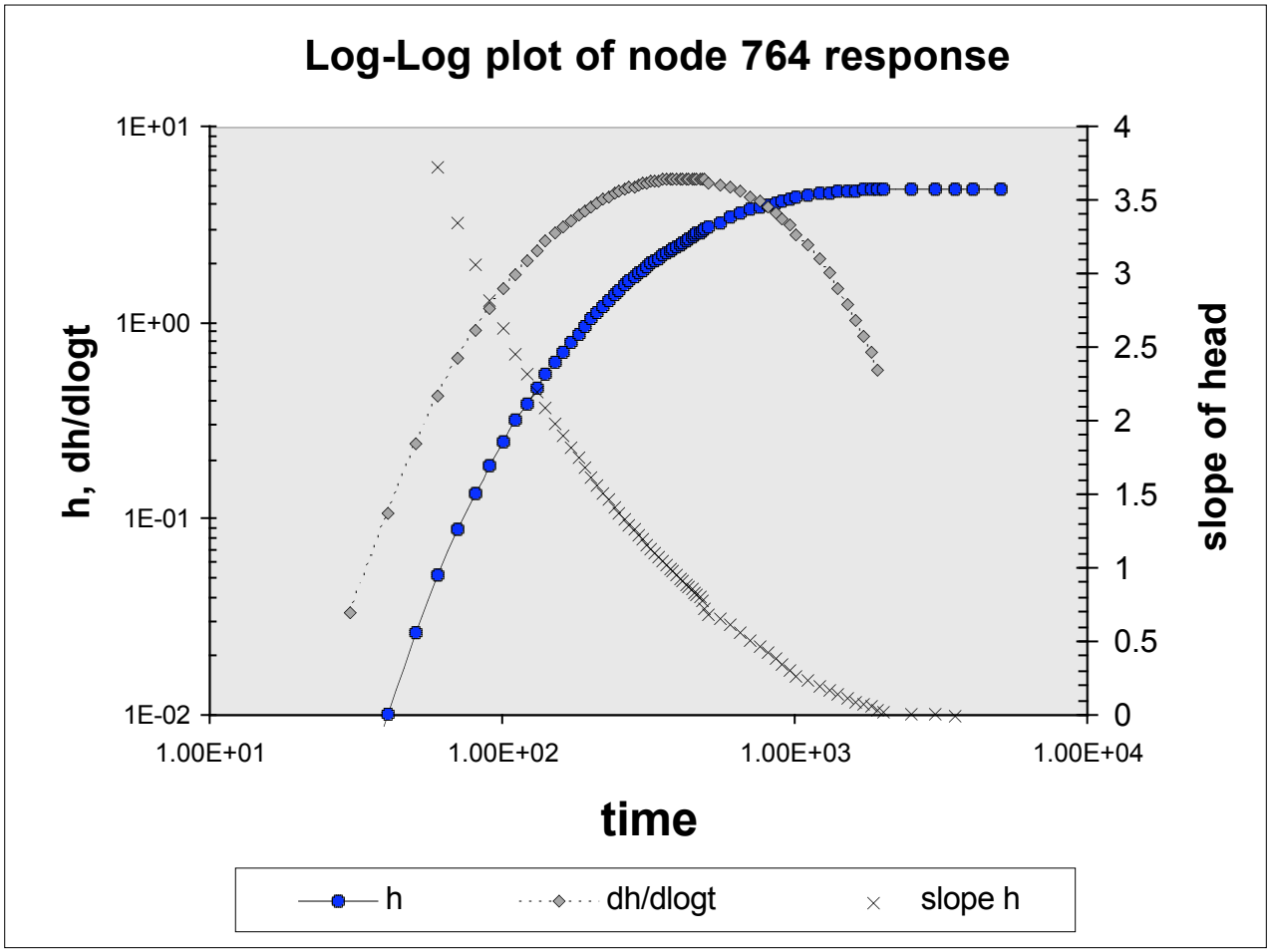
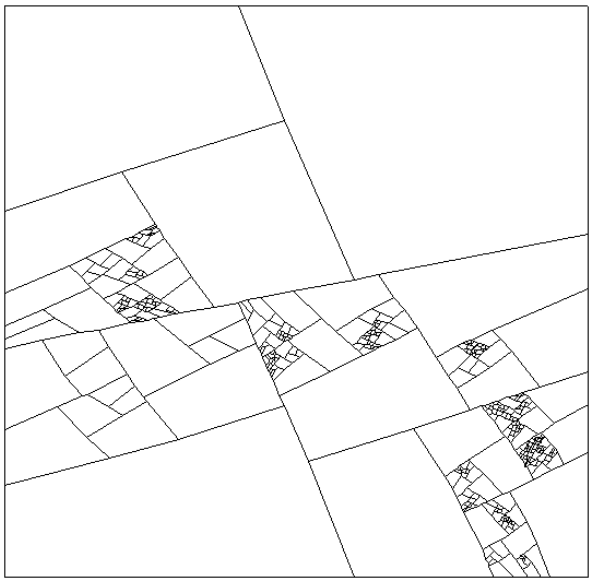


Figure 28: Log-log plot of drawdown and drawdown derivative w.r.t. log time versus time for the observation well halfway between the pumping well and the network boundary. The straight-line section of the drawdown curve has now almost disappeared and the drawdown could now be easily interpreted as a leaky aquifer of a simple Theis-like drawdown with the effects of a boundary at late times. Effectively, the development of the straight-line section does not start before the effects of the boundary are felt.

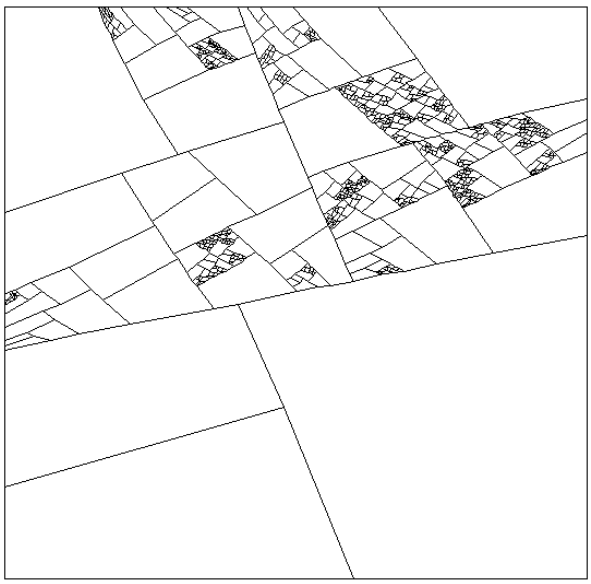
3.3 A stochastic fractal pattern generator

Building on the previous results, it was attempted to generate fractal networks with varying fractal mass dimension. A stochastic method of doing this, only slightly modifying the IFS algorithm, is to specify a "fracturing probability" with which existing fractures of one generation are further subdivided and transformed to form the next generation. The generation at which this random fracturing starts can also be specified. Some numerical properties of these networks will be illustrated next. First, however, it is important to stress that due to the stochastic nature of the algorithm several realisations for a given parameter set should be analysed. Within the scope of this study there was not enough time available to analyse enough networks to allow a statistically valid description. Hence, even though the general trend of results presented here is valid, detailed studies are necessary to obtain more accurate numerical values. Figure 29 shows four different fractal networks that all have the same initial shape and initial fracture geometry but have a varying fracture probability, that was applied for all fractures generated after the first generation. Shown are networks created with a fracture probability of 0.65, 0.70, 0.75 and 0.80.

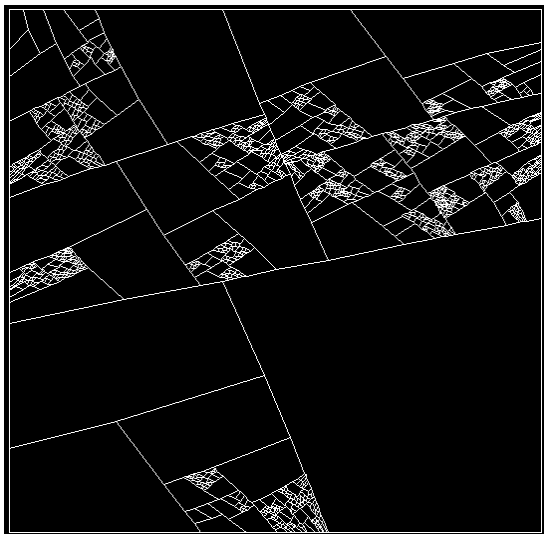
a) 13 fracture generations, prob. 0.65



b) 13 fracture generations, prob. 0.70



c) 13 fracture generations, prob. 0.75



d) 13 fracture generations, prob. 0.80

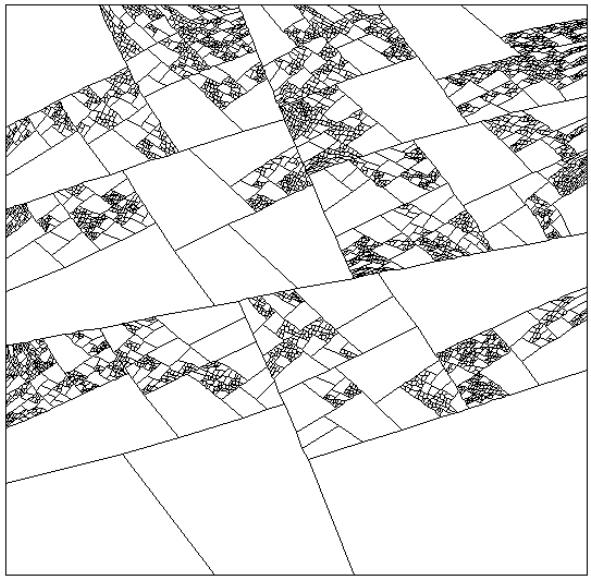


Figure 29: Illustration of four fractal networks with different fractal mass dimensions, generated by varying fracturing probability. Shown are networks with fracturing probabilities of a) 0.65, b) 0.70,
c) 0.75 and d) 0.80. Note that, even though the networks generated already seem to be fairly realistic, better approaches are possible. One example would be to also alter the fracture aperture according to a power law relationship.

3.3.1 The relationship between fracturing probability and fractal and flow dimension

In order to use stochastic fractal network algorithms it is first necessary to understand the relationship between parameters that govern the algorithm, e.g. fracturing probability, and parameters that are to be modelled like fractal dimension and flow dimension (slope of a drawdown curve on a log-log plot). The methodology used here was to

1. generate stochastic networks as shown in figure 29 for various fracture probabilities,
2. analyse the fractal properties of these networks and
3. simulate a pumping test with each network to determine the slope of the drawdown plot.

Various freely available software packages were evaluated for their suitability to give accurate fractal dimensions for the generated networks. Most use a graphical approach, where the binary image of a network is analysed. These codes are mostly based on the box-counting technique described in section 2.3.4. Even though these methods have the advantage of being able to deal with linear features of a graph, they are restricted to two-dimensional networks and a varying fracture aperture is difficult to include and measure. For this project, the most suitable software package found is called “fd3”, written by John Sarraille and Peter DiFalco (john@ishi.csustan.edu). This software accepts co-ordinates of points for any embedding dimension and calculates the capacity, information and correlation fractal dimension. Software was written by the author to convert the raw network data (nodes and bonds) to co-ordinates of fracture mid-points which were then analysed using “fd3”.

The effects of lower cut-off scale

To evaluate the output of the network algorithm, stochastic fractal networks with ten fracture generations were created using different fracture probabilities. Additionally, the fracture generation at which probabilistic fracturing starts was varied. The resulting networks were then analysed using “fd3”. The geometry of the initial fracture is shown in figure 21.
Table 3 shows the data obtained and figure 30 the resulting plots of fractal dimension versus fracturing probability. Figure 30 shows that fracturing probability is strongly correlated with the fractal dimension values. Information dimension seems to be the most suitable fractal dimension measure because it takes into account the effects of discretising line segments into points. A linear regression line is fitted to the information dimension points and the effect of starting probabilistic fracturing at different generations becomes clear from the different slopes exhibited. Different starting generations effectively mimic different lower cut-off scales. Stronger fractal network characteristics are achieved by starting probabilistic fracturing at the earliest generation and this will be used in subsequent investigations. Note that at small fracturing probabilities the number of co-ordinates, as expressed by nodes and bonds in table 3, becomes too small to calculate an accurate fractal dimension, leading to the deviations from a straight line as observed.

a)			starting generation: 1		
fracturing probability	capacity dimension	information dimension	correlation dimension	Number of nodes	bonds
0.60	1.15	1.35	1.32	192	286
0.65	1.23	1.32	1.22	264	394
0.70	1.39	1.35	1.29	376	562
0.75	1.36	1.52	1.53	548	820
0.80	1.64	1.67	1.66	1152	1726
0.85	1.70	1.73	1.73	1484	2224
0.90	1.87	1.84	1.83	2340	3508
0.95	1.96	1.92	1.90	3276	4912
1.00	2.00	1.98	1.96	4098	6145

b)			starting generation: 3		
fracturing probability	capacity dimension	information dimension	correlation dimension	Number of nodes	bonds
0.50	1.18	1.54	1.62	82	121
0.55	1.29	1.26	1.70	160	238
0.60	1.25	1.42	1.42	238	355
0.65	1.34	1.47	1.40	348	520
0.70	1.36	1.53	1.44	502	751
0.75	1.39	1.53	1.51	638	955
0.80	1.64	1.67	1.66	1152	1726
0.85	1.70	1.73	1.73	1484	2224
0.90	1.87	1.84	1.82	2340	3508
0.95	1.96	1.92	1.90	3276	4912
1.00	2.00	1.98	1.96	4098	6145

c)			starting generation: 6		
fracturing probability	capacity dimension	information dimension	correlation dimension	Number of nodes	bonds
0.50	1.73	1.62	1.48	450	673
0.55	1.79	1.66	1.48	584	874
0.60	1.83	1.71	1.55	720	1078
0.65	1.85	1.69	1.61	902	1351
0.70	1.73	1.69	1.57	1124	1684
0.75	1.76	1.73	1.64	1396	2092
0.80	1.83	1.79	1.70	1802	2701
0.85	1.89	1.84	1.76	2224	3334
0.90	1.93	1.89	1.80	2744	4114
0.95	1.97	1.93	1.86	3362	5041
1.00	2.00	1.98	1.91	4098	6145

Table 3: These three tables show the values obtained for capacity dimension, information dimension and correlation dimension for different values of fracturing probability. Table a) shows the results obtained where probabilistic fracturing starts at the first fracture generation, b) probabilistic fracturing starts at the third generation and c) probabilistic fracturing starts at the sixths generation.

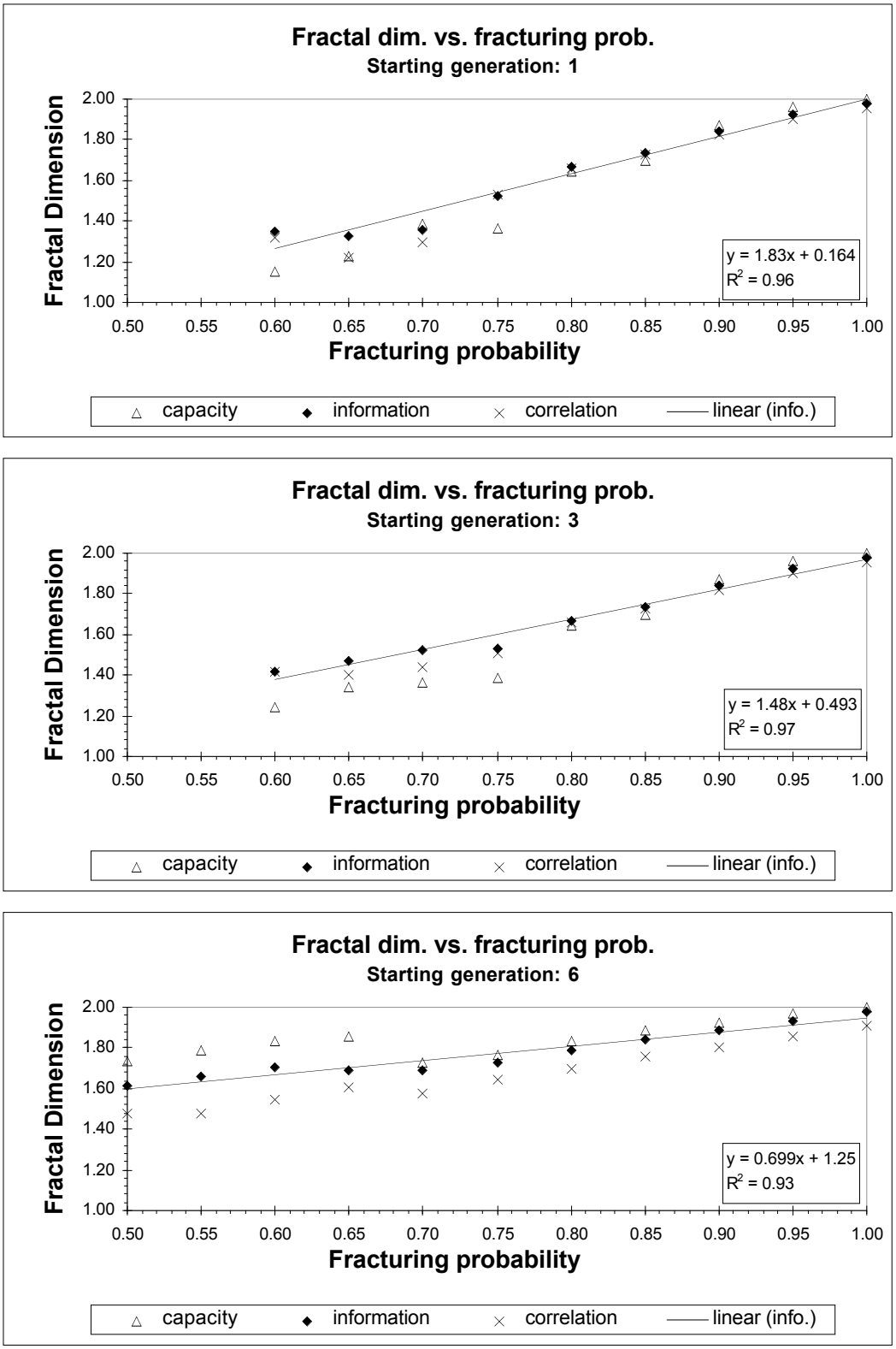


Figure 30: These three plots correspond to the data in table 3. A linear regression line is fitted to the information dimension data points. See main text for discussion.

Different slopes of pumping response versus fractal dimension

The same procedure as used above was now applied to networks with 14 fracture generations. This should enhance the fractal properties of the generated network and enable the determination of a fractal pumping response. The results are shown in table 4 and figure 31.

The relationship between fracturing probability and slope of pumping response							
fracturing	capacity	information	correlation	Slope		Number of	
probability	dimension	dimension	dimension	1-delta	delta	nodes	bonds
0.500	1.00	1.03	0.81	0.32	0.68	84	124
0.525	1.00	1.03	0.81	0.33	0.67	84	124
0.550	1.00	1.03	0.81	0.32	0.68	206	307
0.575	0.86	0.98	0.80	0.26	0.74	206	307
0.600	0.85	1.16	1.09	0.24	0.76	230	343
0.625	1.03	1.17	1.09	0.24	0.76	528	790
0.650	1.14	1.16	1.02	0.31	0.69	706	1057
0.675	1.15	1.26	1.20	0.27	0.73	1042	1561
0.700	1.24	1.33	1.27	0.23	0.77	1462	2191
0.725	1.29	1.36	1.30	0.16	0.84	2070	3103
0.750	1.39	1.39	1.35	0.23	0.77	3006	4507
0.775	1.46	1.48	1.44	0.1	0.9	3998	5995
0.800	1.52	1.57	1.55	0.2	0.8	7610	11413

Table 4: Tabulated values of fractal dimension, slope of pumping response, network size and fracturing probability for a network with 14 fracture generations. The discrepancy between the number of nodes and elements for the same low fracture probabilities between this table and before is due to the elimination of nodes as explained in appendix B.

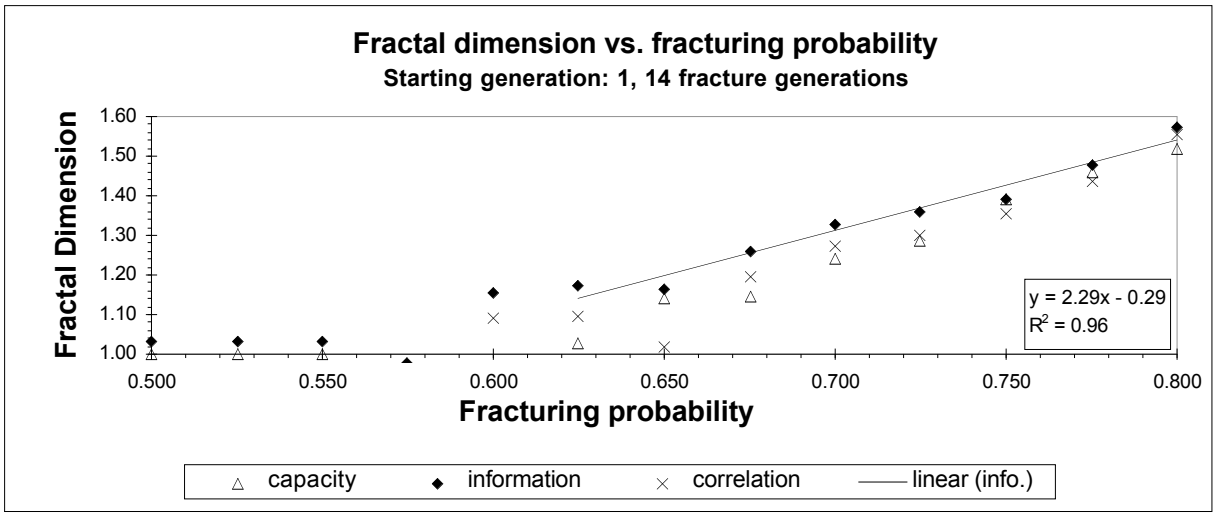


Figure 31: Similar to figure 30, this plot shows the correlation between various fractal dimension measures as determined by the package fd3 and fracturing probability. Again the small number of points at low fracturing probabilities causes a deviation from a linear relationship.

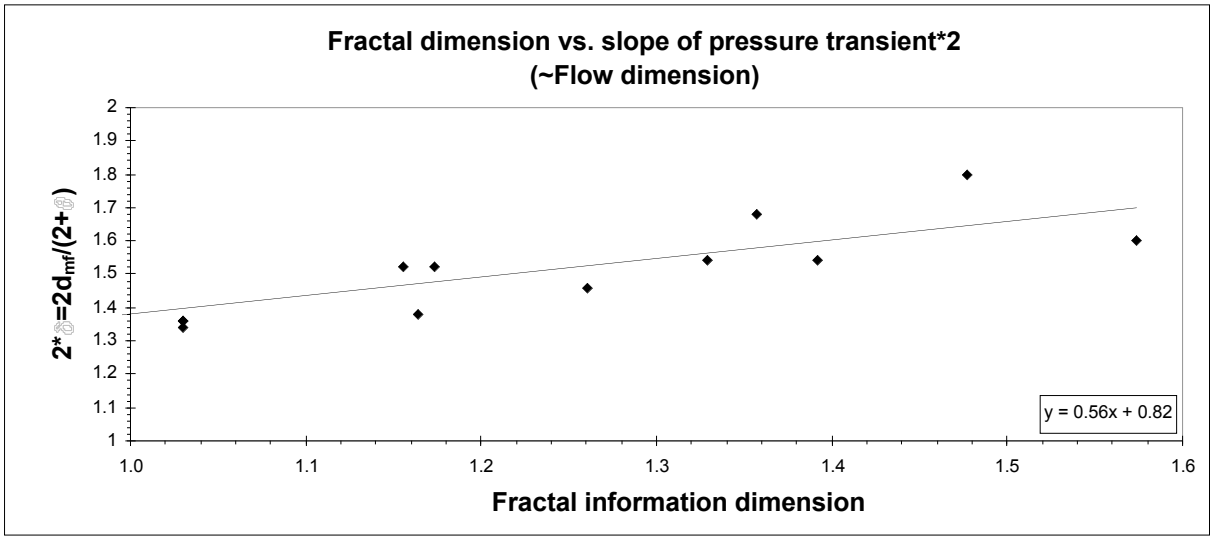


Figure 32: This plot shows the relationship between the information dimension of generated networks and the slope of the pumping well pressure transient, here plotted as 2*δ, which is equivalent to the flow dimension if θ=0 (c.f. equation 26). A linear trend can be observed, although the random nature of the networks and difficulties determining the slope of the pumping response cause a large variability.

4. Inversion modelling using simulated annealing

4.1 Introduction to simulated annealing

This part of the study shows how inverse modelling from pressure transit data might be used to study fractal fracture networks. The numerical technique of *simulated annealing* has been shown to be suitable for optimisation problems of large scale, in particular for problems where the search for a desired global minimum of a function is obscured among many local extrema (van Laarhoven and Aarts, 1987). The most often cited example is the approximation of a solution for the famous *travelling salesman problem* of finding an optimal routing for a tour visiting each of N cities in turn, which is intractable to solve by other methods in reasonable time (i.e. it is a "NP-hard" problem). The application and effectiveness of the simulated annealing algorithm on this problem are illustrated in figure 33.

In the remaining part of this section the general background for the simulated annealing stochastic inversion method is introduced, followed by an account on how this method can be applied to inverse modelling of pumping test data.

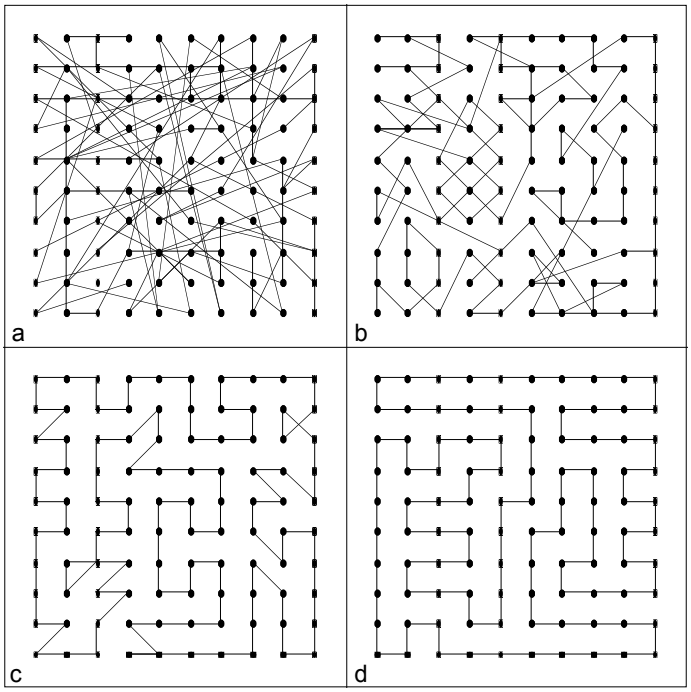


Figure 33: Configurations of a 100-city travelling salesman problem obtained after 0, 68, 92 and 123 steps of the algorithm. The initial tour looks very chaotic: (a) (high entropy). Gradually, the tour becomes less chaotic: (b) and (c) (the entropy decreases). The final tour (d) shows a highly structured and regular pattern with

minimum cost and entropy. After van Laarhoven and Aarts, 1987.

4.2 Origin of the simulated annealing algorithm

In its first description (Metropolis et al., 1953), the simulated annealing algorithm is derived from an analogy with thermodynamics, specifically with the way that liquids freeze and crystallise, or metals cool and anneal. At high temperatures, the thermal energy of molecules or atoms in a melt is high enough for them to move around freely with respect to one another. If the melt is cooled sufficiently slowly rather than quenched, thermal mobility is not lost faster than the ability of atoms to rearrange themselves into a lower energy configuration. In the ideal case a state of minimum energy is reached in the form of a crystal or ordered solid. It is important to allow *slow* cooling, giving enough time for the redistribution of atoms as they lose mobility so that the total system energy is minimised.

Metropolis et al. (1953) used a simulation that was based on a probability distribution for the range of energies called the Boltzmann distribution. At each temperature value T, the solid is allowed to reach *thermal equilibrium*, characterised by a probability of being in a state with energy E:

$$\Pr\{\mathbf{E} = E\} \quad \exp^{\left(\frac{E}{k_B T}\right)} \qquad (34)$$

where T is the temperature and k_B the Boltzmann constant. As the temperature decreases, the Boltzmann distribution concentrates on the states with lowest system energy and as $T \to 0$, only the minimum energy states have a non-zero probability of occurrence.

Kirkpatrick et al. (1983) realised that there exists an analogy between minimising the cost function of a combinatorial problem and the slow cooling of a solid and that by applying the annealing concept to discrete minimisation problems a near optimal solution can be found. Asymptotically, simulated annealing converges to the global minimum (van Laarhoven and Aarts, 1987).

The simulated annealing works by taking the system configuration to represent the molecular state distribution. The cost or energy function is analogous to the free energy state and the set of free parameters (configuration) is analogous to the arrangement of molecules. To mimic the behaviour of these molecules, the parameter configuration is changed at each step of the annealing process by a small perturbation. Typically, if this change results in a more stable (i.e. lower energy) system, the new configuration is accepted. This process is repeated until an energy equilibrium is reached.

4.3 Description of the simulated annealing algorithm

4.3.1 The hiker analogy

The search for parameters for a theoretical model so that it fits a set of observation data is analogous to the problem of a hiker who has been dropped into a very hilly region where he or she is expected to find the lowest point (Mauldon et al., 1993). The hiker can remember his last previous position (configuration) and altitude (energy) and can predict how far up or down a proposed step will take him, but he cannot see farther than this next step. A simple search strategy for the hiker would be to always take a step downhill which is equivalent to often employed gradient method. However, it is unlikely that this is the desired deepest valley (the global minimum).

A better search strategy would always allow the hiker to choose a proposed downhill step and sometimes allow him an uphill step. This would allow him to jump out of a local minimum. The hiker may take an uphill step based on a random process with a decreasing probability of taking large steps uphill. This, in essence, is the how the simulated annealing method works.

4.3.2 The Metropolis criterion

The following test is applied in determining a transition to the next configuration and is known as the Metropolis algorithm or criterion (Metropolis et al, 1953). Generally, given a fixed temperature, this test always allows a transition to a configuration if system energy is decreased and sometimes allows a transition to a configuration with higher energy. The

probability of a transition to a higher energy state decreases exponentially with the size of the energy increase. This is the stochastic relaxation step that allows simulated annealing to search the space of possible configurations without always converging to the nearest local minimum.

For the **n-1st** configuration $\mathbf{X_{n-1}}$ and the **nth** configuration $\mathbf{X_n}$ with energies $\mathbf{E_{n-1}}$ and $\mathbf{E_n}$ respectively, the transition probability at system temperature **T** is given by

$$\mathbf{Pr}(X_{n-1} \rightarrow X_n) = \begin{cases} 1 & \text{if } E_n - E_{n-1} < 0 \text{ or} \\ \exp(-(E_n - E_{n-1})/T) & \text{if } E_n - E_{n-1} > 0 \end{cases}$$

If the temperature **T** is held constant, the system approaches thermal equilibrium and the probability distribution for a configuration with energy **E** approaches the Boltzmann probability as given by equation (34).

Typically, the annealing temperature is held fixed for a prescribed number of transitions. This temperature schedule $\{(T_k, L_k);\ k = 1, 2, \ldots\}$ must be input by the user and is usually unique to the optimisation problem. However, there are general recommendations for the design of temperature schedules (van Laarhoven and Aarts, 1987).

In general, simulated annealing can be described by the following algorithm:

```
do parameter initialisation
initialise temperature  T = T[0]

 repeat until E  near equilibrium
 {
    while temperature schedule not exhausted
        generate configuration j from configuration i  by small perturbation
        calculate  ΔE_ij = E_j - E_i
        if ΔE_ij ≤ 0 accept j
        else if exp(-ΔE_ij /T) > Z accept j
        where Z is a random variate from a uniform U(0,1) distribution.
        if accept j,
              update current model to configuration j
              increment # acceptances at temperature T
        if  # acceptances at temperature T is L_k
              lower the temperature:  T = T[k +1]
}
```

It is important to realise that the original algorithm can only be applied to *combinatorial minimisation*. There is a objective function to be minimised, but the space over which that function is defined is not the usual N-dimensional space of N continuously variable parameters. Instead, the configuration space consists of a *discrete,* but very large set. The number of elements in the configuration space is factorially large, so that they cannot be searched exhaustively. Press et al. (1986) describe refinements to the algorithm to prevent a situation where , if the search region consists of a long narrow valley, random steps will most likely lead to an uphill direction.

To make use of the Metropolis algorithm for other than thermodynamic systems it is necessary to provide the following elements (Press et al., 1986):

1. A description of possible system configurations
2. A generator of random changes in the configuration; these changes are the "options" presented to the system
3. An objective function E (analogous to energy) whose minimisation is the goal of the algorithm
4. A control parameter T (analogous to temperature) and an *annealing schedule* which tells how it is lowered from high to low values, e.g. after how many random changes of a configuration or after how many downward steps the value of T is lowered, and how large that step is.

4.4 Adaptation for flow in fractured media

The use of the simulated annealing method is relatively new in the field of hydrogeology. It was first used in hydrology to find a groundwater management strategy (Dougherty and Marryott, 1991). The main work concerning the use of simulated annealing for inverse modelling of pumping test data has been done by Mauldon et al., 1993; Najita, 1994 and Datta-Gupta et al., 1995 at Lawrence Berkeley Laboratory (LBL). Julie Najita (LBL) provided the code for inverse modelling using simulated annealing used in this study. This code was only slightly modified by the author to run on different computer systems (it was made ANSI

compliant) and to output data files in a suitable format. The approach described below is entirely that of LBL and the kind help of Julie Najita is acknowledged here.
The original simulated annealing algorithm is designed to work on a discrete configuration space. Hence, a problem involving a system of discrete fractures is amenable to study with this method whereas a homogenous porous medium is not without modification (Press et al., 1986).

Fluid flow that takes place in fractured rock is often observed to be confined to a poorly connected network of fractures. Typically, fracture maps from drift walls and outcrops, and fracture logs from boreholes can provide a fracture geometry that may control flow. However, interference well test and tracer test data have shown that only some of the fractures are hydraulically conductive (Olsson, 1992). This may be due to poor connectivity, in-situ stress fields and in-filling materials.

Simulated annealing is used to find fracture network models that match observed pressure transient curves from pumping tests.

To apply the simulated annealing algorithm to the problem of groundwater flow in fractured media, all four elements of the algorithm as described above have to be developed.

4.4.1 A description of the possible system configuration

Possible system configurations are described by conceptualising a fracture system as a lattice of fracture elements. Fracture elements are modelled by lattice bonds along which flow takes place and is similar to a finite element mesh. The original configuration, a generic network, is called a ”template” which is then modified later to approximate the observed pumping test data. Ideally, the template should have a geometry that does not constrain the degrees of freedom of connectivity, i.e. most possible flow configurations should be possible using a given template. It was shown by Mauldon et al. (1993) that the exact orientation of this template with respect to the flow system does not greatly influence the results obtained, as long as the grid is connected enough to allow a reasonable variety of system configurations.

4.4.2 The perturbation step

The approach as chosen by workers at LBL is to perturb fracture network configurations of the original template at each iteration by changing the aperture size for a cluster of network elements, where the maximum number of apertures is specified. The new cluster aperture is determined by randomly selecting a value from a list specified. Fluid flow through the new fracture system is simulated using a Galerkin finite element method (TRINET, Karasaki, 1987). Details of the fracture network model and techniques can be found in the section on technical implementation section and in the software supplement that is provided with this thesis in electronic form.

The value of the objective function E is obtained by comparing the simulated pressure transients with the observed values. System energy is defined as the sum of differences squared between simulated and observed pressure for each observation point and time step. Even though this method is not ideal (it gives more weight to late time, large drawdown values), the author found it to be better than a percentage objective function of the form $\left| \frac{observed - simulated}{observed} \right|$ because the latter gives too much weight to early time, very small drawdowns.

4.4.3 The cooling schedule

Several different approaches to develop a cooling schedule for a given annealing problem can be found in van Laarhoven and Aarts (1987). The method used here is to proceed to the next (lower) temperature when a given number of downhill steps are accepted. The algorithm is terminated when the temperature schedule reaches the end or when a specified number of acceptances or rejections of a new configuration has been reached.

4.5 Technical implementation

4.5.1 Fracture network model

The used inverse code is capable of solving 3-D problems, but for the purpose of this project only 2-D models were used due to constraints on computing power and memory requirements. The model is constrained to constant flux well tests and in the 2-D case a vertical fracture plane with unit thickness is used.

- The first part of the software package, which was added by the author, consists of a fracture network mesh generator as described in section 3.1. This generates input files that consist of an element and a node file. The element file specifies fracture segments by two node numbers and the original transmissivity, aperture (width) and a storage term. Note that the storage term used here is of the form $Storage = S_S \cdot length \cdot aperture \cdot unit\ thickness$. This is dimensionally correct but not standard use. The reason for this the conceptual problem associated with fracture porosity. The node file specifies the spatial co-ordinates for each node and associated initial head boundary type and values. The mesh generator uses an iterated function system approach which is capable of generating complex, fractal fracture networks in addition to regular 2-D lattice templates. The mesh generator is discussed fully in the section on fractal fracture networks. Details of the exact input and output file formats can be found on the included disks.
- Second, well test boundary conditions are imposed on the mesh with separate input files. This allows the change of pumping rates at specified times.
- Third, a finite element code, TRINET (Karasaki, 1987), is used to solve for the head distribution at each time step in the mesh by solving the transient flow equation for each element:

$$K\frac{\partial^2 h}{\partial x^2} = S_S\frac{\partial h}{\partial t} \quad \text{..... (35)}$$

where K is the fracture hydraulic conductivity, x is the length along the direction of the fracture, S_S is the specific storage of the fracture, and h is hydraulic head. It is assumed that

K is constant and that flow through fractures is laminar. It is further assumed that the fracture aperture is large enough so that Poiseuille's law (or cubic law) holds:

$$T = \frac{\rho g w^3}{12\mu} \quad \text{..... (36)}$$

where T is the element transmissivity, ρ is the density of water, μ the dynamic viscosity of water, g the gravitational acceleration and w the fracture aperture. Consistent units are assumed throughout.

4.5.2 Energy calculation

After each iteration step, the well tests are simulated on the fracture network and the calculated drawdown values are compared with observed (input) values. The system configuration energy is defined as energy

$$E = \sum_{k=1}^{Nw} \sum_{i=1}^{nk} \left(s_{ik}^{obs.} - s_{ik}^{calc.} \right)^2 \quad \text{..... (37)}$$

where $s_{ik}^{obs.}$ = observed drawdown at i^{th} observed time for k^{th} well

$s_{ik}^{calc.}$ = calculated drawdown at i^{th} calculated time for k^{th} well

N_w = number of wells

n_k = number of observations at k^{th} well.

If times for observation and calculated times do not coincide, calculated drawdowns are determined by linear interpolation between observed times.

4.5.3 Cluster variable aperture perturbation

Instead of just changing one fracture segment per iteration, the code allows to specify a "cluster size". During each iteration step, a number of *connected* fractures up to this cluster size are selected. Then, an aperture value is randomly chosen from a list supplied by a

parameter file and assigned to the selected fracture segments. Then the flow equations are solved, the system energy re-calculated and the next iteration process started. This can speed up the parameter search because the hydraulic parameters of the aquifer might be spatially correlated and some areas of the network are more important in defining the pumping the response than others (e.g. near the pumping well).

4.6 Simulated annealing evaluated on simple networks

In order to evaluate the efficiency of inverse modelling through simulated annealing a first run was conducted on pumping data generated from a relatively simple network. The grid used for forward modelling consists of 21 by 21 nodes, defining a square grid. Each bond connecting two nodes is assigned one of two fracture apertures. A central, circular region, indicated by a light colour in figure 34, is assigned a small fracture aperture, simulating a low transmissivity region. The outer region is assigned a higher value. One pumping well and 7 observation wells are located on nodes around the circular inner region. A pumping test is then simulated and the resulting pressure transient data fed into the inverse part of the code, using as "template" a grid of the same size as the original input network, starting with a uniform aperture. A cooling schedule was chosen empirically by trial and error (see van Laarhoven and Aarts, 1987, for more sophisticated methods). The inversion code was allowed to assign one of two apertures to each bond. After ~12000 iterations the output of the code is shown in figure 35.
Several key features of using simulated annealing to inverse model pumping test data can be identified by comparing input and output networks:

1. The output transmissivity distribution is very similar in visual appearance to the original, although not exactly the same. This shows that simulated annealing will find a minimum that is very *close* to the global minimum under the right conditions.
2. Towards the outer (constant head) boundary, the effects of transmissivity on the drawdown of observation wells located around the centre are very small, explaining the larger differences between input and output towards the boundary. This is an important feature, suggesting to design grids with boundaries well outside the region of interest. It also shows the need for as many observation data as possible.

3. The low transmissivity region between the pumping well and the opposite corner of the grid “shields” the pressure transient from affecting certain parts of the network, explaining larger deviations in that part of the grid.

The main purpose of this first inverse run was simply to test that the code works and this can be confirmed. It is also interesting to monitor the evolution of system-energy (misfit between model and input drawdowns) as the number of iterations increases. Figure 36 illustrates that due to the stochastic nature of the algorithm there are considerable high frequency variations in the misfit as it evolves during iterating the simulated annealing algorithm. However, as the cooling schedule proceeds, the decreasing trend in misfit is clearly seen.

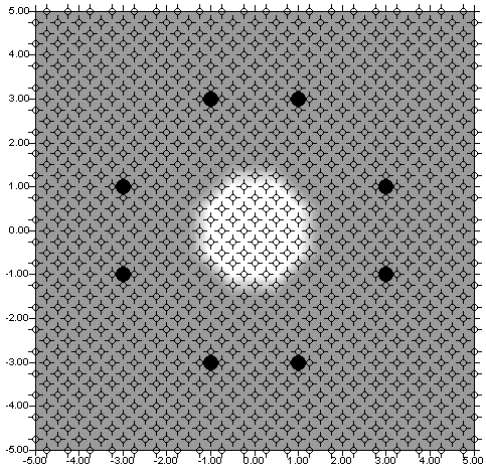


Figure 34: This figure shows the input grid used to generate synthetic drawdown transients. The pumping well is situated in the bottom left corner. The other observation wells are indicated by black circles. Crosses indicate the midpoints of bonds. The light coloured region in the centre has a transmissivity (as obtained from the fracture aperture by using the cubic law) two orders of magnitude lower than the outer region.

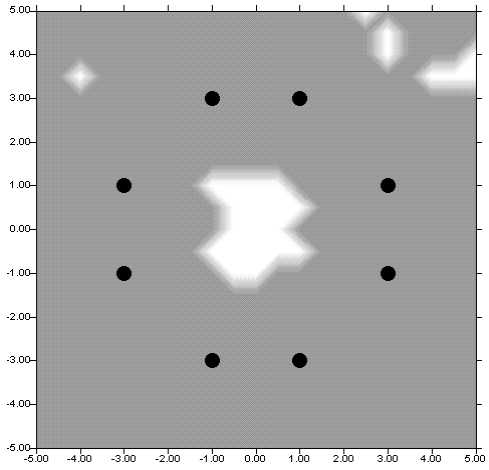


Figure 35: This figure shows the transmissivity distribution obtained by inversion of the synthetic pumping test data.

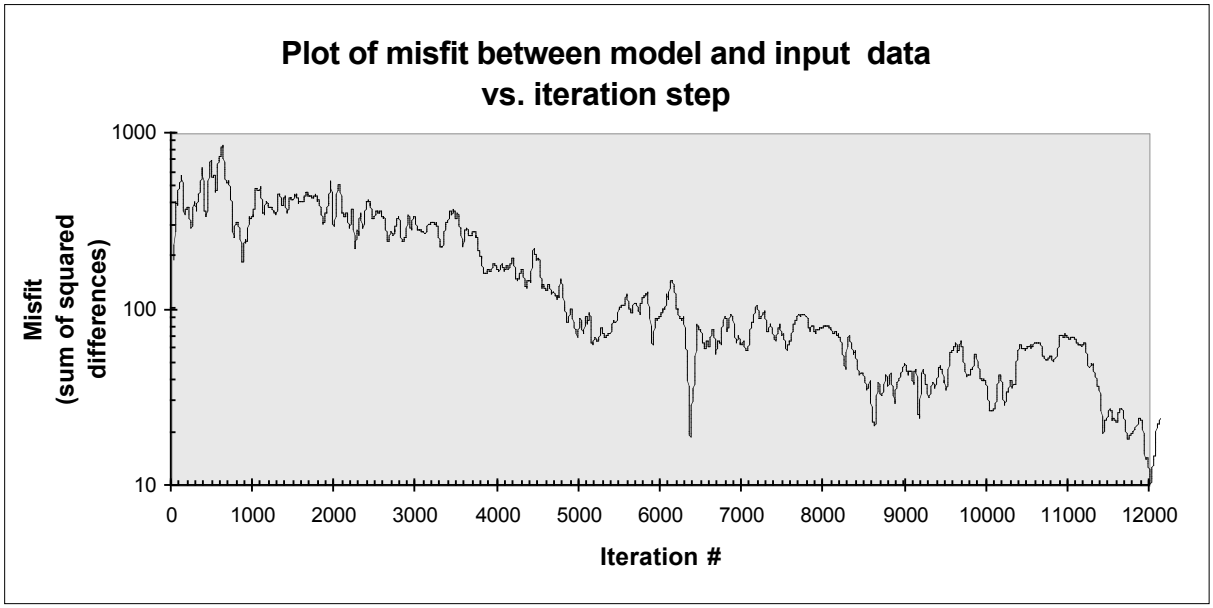


Figure 36: This plot shows the gradual improvement in fit between model and input drawdown data for the model shown in figure 30. Note that data are plotted on a log-linear scale. A statistical analysis of the variation could be used to design appropriate cooling schedules (see van Laarhoven and Aarts, 1987).

4.7 Inverse modelling from fractal pressure transient data

It has been shown in section 3 how it is possible to generate synthetic pressure transient data showing a non-integer or "fractal" behaviour by creating networks with fractal geometric properties and simulating a pumping test. The main aim of many *inverse* modelling approaches in hydrogeology is of course to predict hydraulic properties in the sub-surface environment from a limited set of data, e.g. pumping tests. Hence, if aquifers have a fractal structure that determines the hydraulic behaviour, it is necessary to evaluate to which extent it is possible to quantify fractal parameters from pumping tests. The question thus posed is "Does inversion of 'fractal' pressure transient curves lead to networks with fractal geometric properties?". This question was addressed as part of this study. It is important to realise, however, that the results obtained are only strictly valid for the set of boundary conditions that are used to generate synthetic data in this study. In other words, it is not valid to infer if or if not *all* pressure transients that can be described by non-integer flow models are actually caused by an underlying fractal structure of the aquifer. This study can only be a first step towards are more comprehensive study of fractal aquifers.

The procedure used to address the posed question can be subdivided into the following steps:

1. Generating a synthetic fractal fracture network.
2. Generation of synthetic pressure transient data from this network.
3. Specifying a template that is used for inversion.
4. Generating new networks by inversion that approximate the pumping response.
5. Comparing input and output pressure transients.
6. Obtaining an 'average' network representation.
7. A fractal characterisation of networks

These steps will now be described in detail.

1. **Generating a fractal fracture network**.

   Using the methods and computer code discussed in section 3, synthetic pressure transient data with fractal characteristics are generated by simulation of fractal networks. As a first step, a deterministic fractal pattern is used because its properties are known. For this part

of the study a compromise had to found between large fractal networks with many fractal generations that increase the fractal characteristics of flow behaviour, and the available computing power. The network used here is a modified Sierpinski gasket with nine fracture generations (see figure 37).

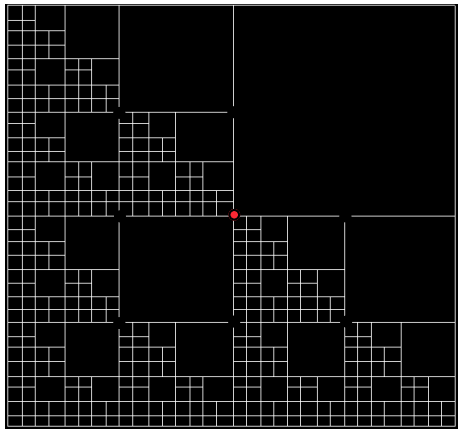


Figure 37: This figure shows the fractal network that was used to generate synthetic pumping test data. The network is a modified Sierpinski gasket as described before, with nine fracture generations. The pumping well is located in the centre, surrounded by seven observation wells. The outer boundary is defined to be of constant head. Network parameters used are shown in table 5.

Type of grid:	**Sierpinski (d_{mf}=1.58)**
Number of fracture generations	9
Assigned storage coefficient	5E-02
Assigned aperture	4.9647E-05
Transmissivity (from cubic law)	1.0E-07
Spatial extent of x/y axes	-5 to +5
Number of nodes	441
Number of elements (bonds)	840
Pumping rate	1.0E-5
Pumping time	300
Number of time steps	66

Table 5: This table summarises the important parameters used to generate the synthetic pressure transient data. Note that no units are specified, self-consistent units are assumed throughout. Note also, that an unrealistically high value for

storage was chosen in order to delay the effects of the constant head boundary.

2. **Generation of synthetic pressure transient data.**

On this grid, a pumping test is simulated to obtain the pressure transients for the pumping well and seven observation points. The synthetic pressure transient data obtained are shown in figure 38. Again it was necessary to find a balance between many time steps and the available computing power. It can be seen from the drawdown plot that the pumping well exhibits a straight-line behaviour, as predicted by fractal flow theory. The limitation on the number of time steps results in observation well responses that are only parallel to the pumping well curve at late times. This is a serious limitation but it is likely to be also the case for many real pumping tests in low permeability rock.

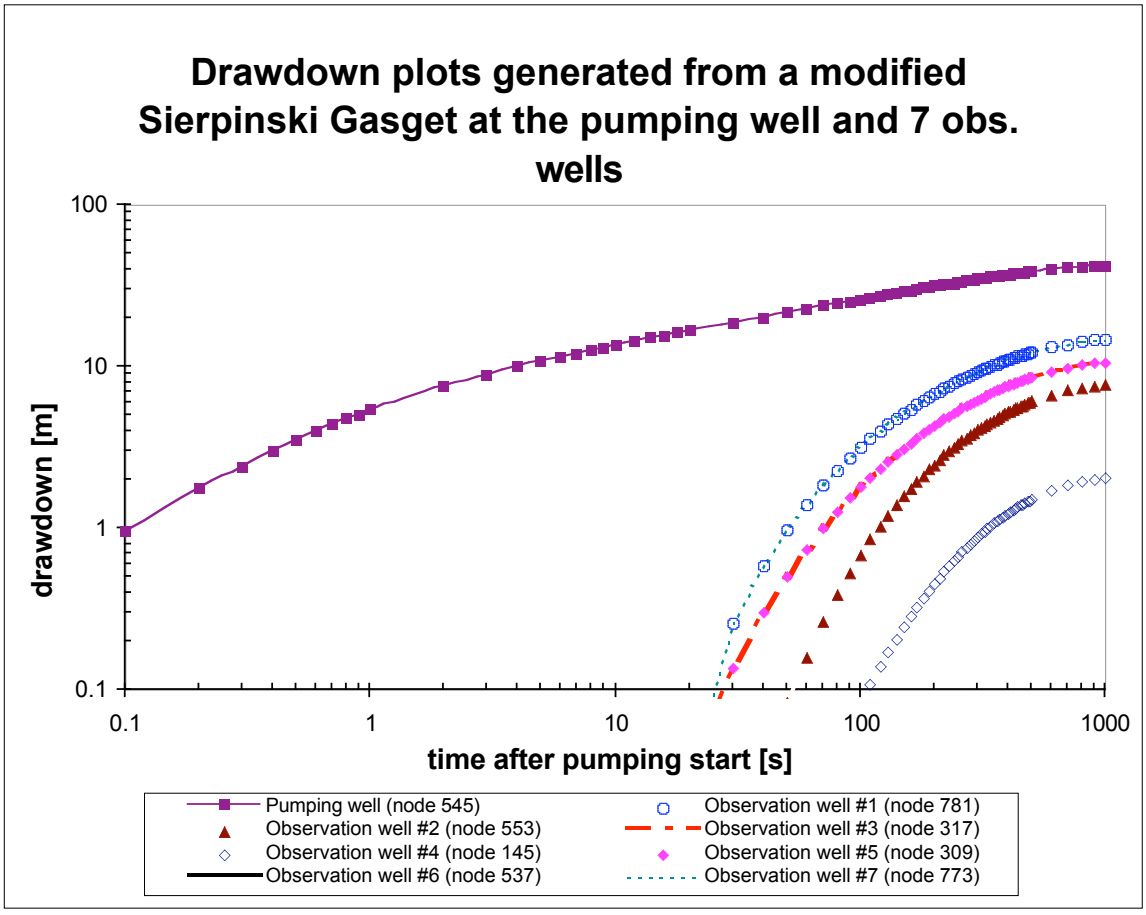


Figure 38: This log-log plot of drawdown versus time shows the pressure transient data obtained from the pumping well and the seven observation wells.

3. **The template used for inversion.**

The obtained drawdown data are then prepared for input into the simulated annealing inverse code. The code has no knowledge of the underlying network. Instead, a regular 2-D "template" grid, as shown in figure 34, is used. The algorithm then modifies individual bond apertures. For this part of the study only two aperture values were allowed: the original aperture of the input grid and an aperture value several orders of magnitude lower in order to simulate bonds "closed" to flow.

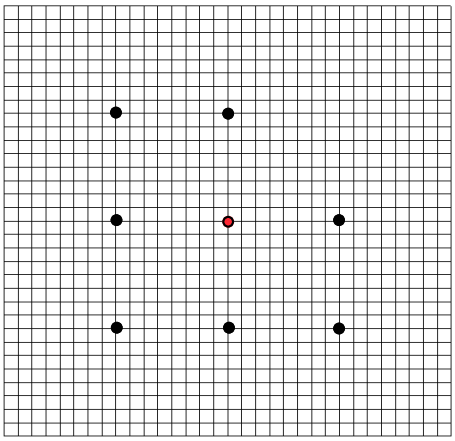
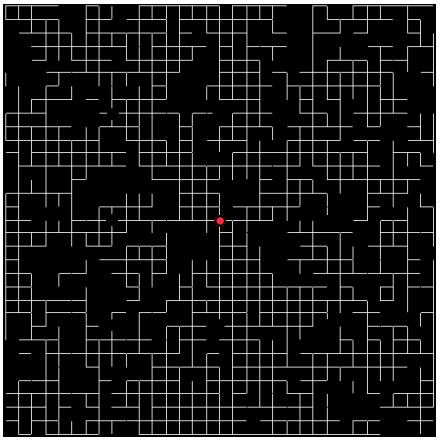


Figure 39: The drawdown data are fed into the inverse modelling code, starting with this 2-D regular grid as "template". Clusters of fracture apertures are then modified to fit the input data.

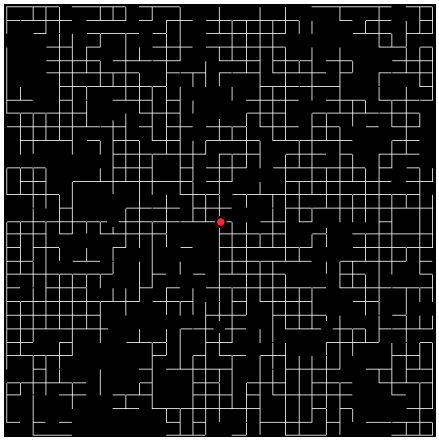
4. **Generating networks by inversion.**

The simulated annealing algorithm is then run several times using the same input data but different random seeds to see how similar different realisations of different generated networks are. Figure 40 shows six networks obtained this way. Each run took approximately 13 hours computing time on the UCL UNIX mainframe (a Sun SPARC with four 248MHz processors), each with ~20,000 iterations. By comparing the original input network with these six realisations it becomes clear that they look very different. On closer inspection, however, several features of the original can still be recognised, e.g. the large top right fragment is represented by very few bonds connecting the pumping well with this region. This can also be seen for other larger blocks.

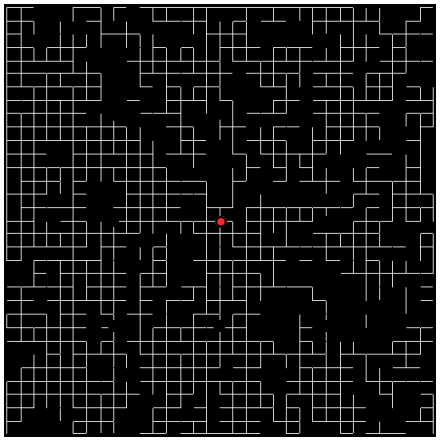
a) Inverse Run #1



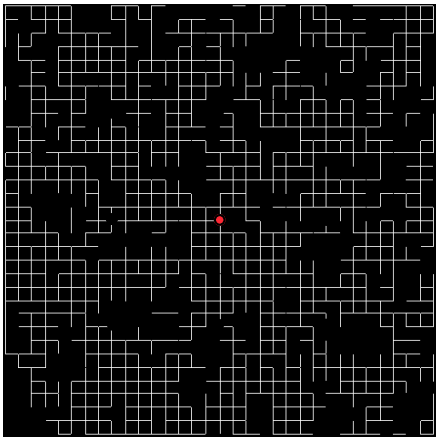
b) Inverse Run #2



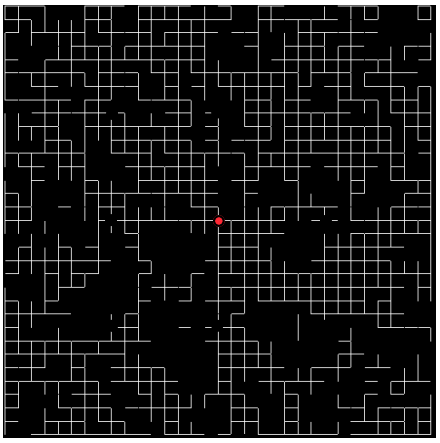
c) Inverse Run #3



d) Inverse Run #4



e) Inverse Run #5



f) Inverse Run #6

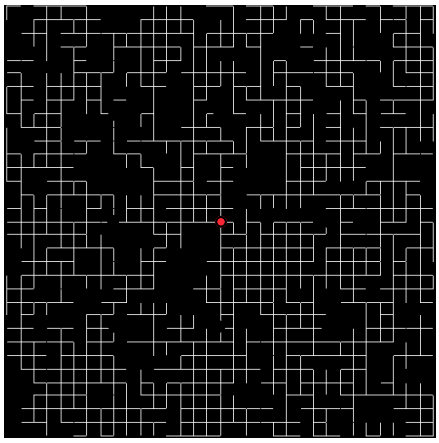


Figure 40: This figure shows six examples of networks obtained by inversion. Note the large variation between networks and the original, representing the amount of uncertainty. Some features of the original input grid can still be recognised, e.g. the large top right fragment in figure 37 is represented by very few bonds connecting the pumping well to this region.

5. **Comparing input and output pressure transients.**

Even though the generated networks seem to be visually very different, their pressure transient response closely resembles that of the original input network. Figure 41 shows drawdown and its derivative with respect to the logarithm of time at the pumping well for the original grid and the six grids obtained by inversion. It can be seen that, apart from inversion run #1, all plot on almost exactly the same curve. The drawdown derivatives show a similarly good fit except, again, inversion run #1. The region where drawdown and derivative approximate straight parallel lines, characteristic for "fractal" flow, can also be clearly seen.

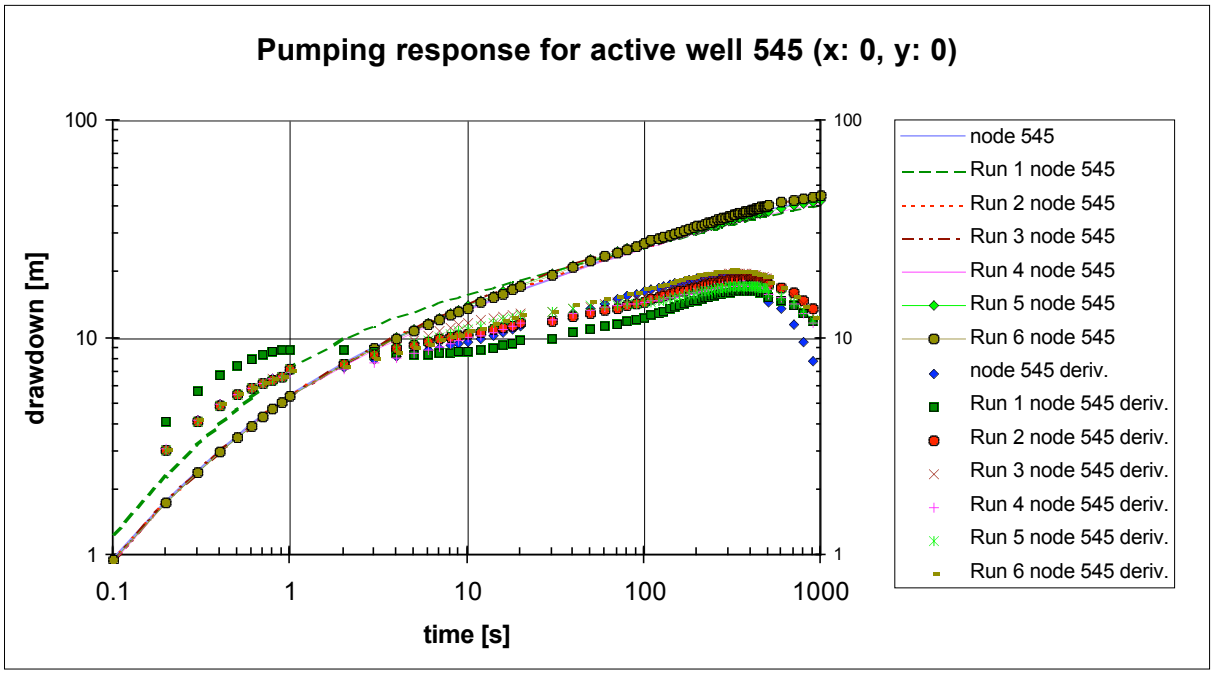


Figure 41: This log-log plot shows the drawdown and drawdown derivative w.r.t. time at the pumping well of the original grid and the six inversion runs.

6. **Obtaining an 'average' network representation**.

Comparing the different network realisations it becomes clear that they share certain characteristics. For example, as pointed out before, the top right quadrant of all networks is only sparsely connected to the pumping well. The similarities and differences between the different networks could be used to derive a measure for uncertainty. This indicates which parts of a given grid are well defined by the available data and where, if a large variation exists, more data need to be collected. Ideally, this should be done using a proper statistical analysis, requiring more than just six inversion runs. One preliminary analysis that was performed on the six grids obtained by inversion was to take the median value of aperture for each individual bond. The resulting 'average' network is shown in figure 42, where red indicates high transmissivity ($T=1.0 \times 10^{-07}$) and blue medium transmissivity ($T=1.0 \times 10^{-12}$). Bonds with a transmissivity of $T=1.0 \times 10^{-16}$ were taken to be closed to flow and are not drawn. This rudimentary method of analysis could be improved by computing the median value of aperture or transmissivity not just for each bond individually but by windowing certain clusters of bonds, i.e. taking into account the neighbourhood of each bond also.

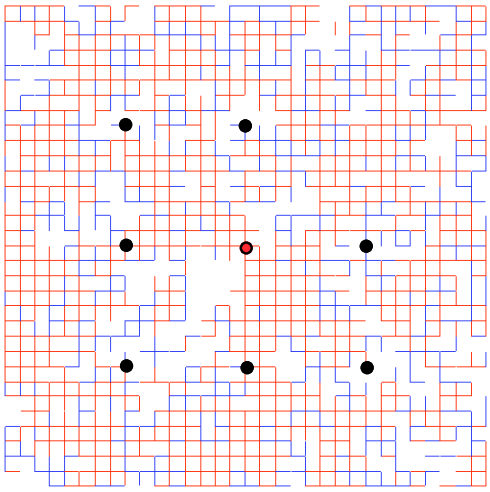


Figure 42: This figure shows an "average" representation of the six output grids. A median value of transmissivity was calculated for each individual bond from the six grids. Red represents a transmissivity value of 1.0×10^{-07}, blue of 1.0×10^{-12} and bonds not drawn of 1.0×10^{-16}.

7. **A fractal characterisation of networks**.

It was shown above that, even though the six network realisations obtained by inversion look very different to the input grid, they approximate the pumping response of the original grid very well. Now an attempt was made to quantify the extent to which the original grid and the output grids share the same *fractal* characteristics. In general, the inverse code will produce networks where the bonds can be assigned more than just two different aperture values. For example, if data from real pumping tests are analysed, the range of transmissivity values is not known a priori. Instead, the user will provide a list of possible transmissivities for the code. This means that normal box-counting methods to determine a fractal mass dimension for the network would fail. Other reasons that a different method of fractal characterisation should be used include the difficulty in extending box-counting image analysis to 3-D and the possible influence of the template geometry on the box-counting method. Here, a newly developed method is used to obtain a fractal dimension that is related to the hydraulic behaviour of a network. Computer code was developed to compute the number of discrete fracture intersections of a circle centred on the pumping well (the centre of the network) and the total cumulative fracture volume (fracture length x fracture aperture x unit thickness) inside that circle for different radii. This combines a measure for the connectedness of a network at different distances around the pumping well (the number of fracture intersections) with the total fracture volume, a measure that more resembles the traditional fractal mass dimension. Note that the fractal exponent obtained depends on the numerical values used for the network such as spatial extent and fracture apertures. Hence, it is difficult to directly link the obtained value to actual flow equations but it can be used to compare different networks. Note also the information needed to compute this fractal measure cannot be easily obtained from field measurements, hence its use is probably constrained to synthetic networks.

A plot of this new fractal measure for the original input network is shown in figure 43. it can be seen that it is possible to fit a straight line to the resulting plot, hence the method manages to capture the fractal properties of the network.

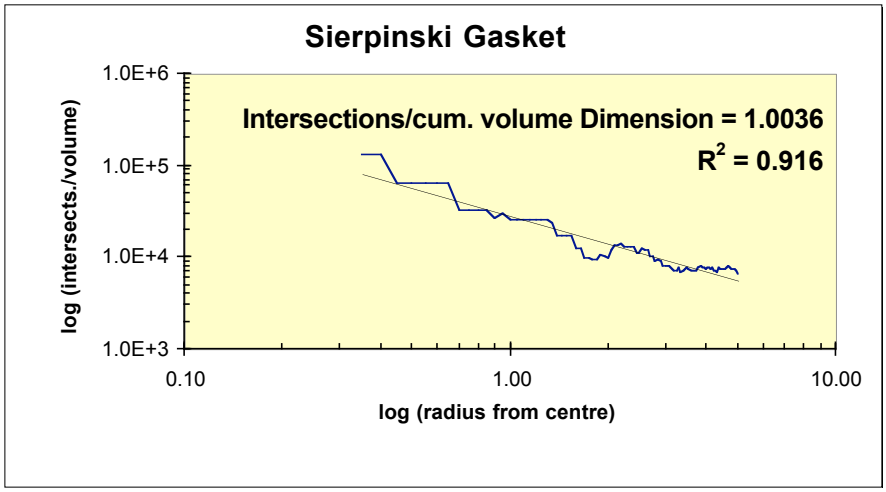


Figure 43: This figure shows the radial variation of the fractal measure "fracture intersections / cumulative fracture volume". Even though the resulting log-log plot shows the discrete nature of the underlying grid, a straight line can be fitted by linear regression and a fractal exponent is obtained from the slope.

The same method of measurement is now applied to the six networks obtained by inversion in order to compare the fractal characteristics with the input grid. The results are shown in figure 44. All six curves can also be approximated with a straight line and the general appearance is remarkably similar to the input network graph of figure 43. It can be seen that the more random nature of the networks results in a slightly smoother curve. By comparing the actual fractal measures obtained it can be seen that all are very similar to that obtained from the input network: five out of six are less than 1.1. Another feature that suggests that this new fractal measure is useful to characterise the hydraulic behaviour of a network can be seen by comparing the fractal dimension and the flow behaviour of inverse run #1. This network produces the largest deviation from the pumping response of the original network (see figure 41) and shows the largest deviation in the fractal measurement. This probably means that the simulated annealing algorithm stopped at a local minimum, again showing the need to run the algorithm several times with different random seeds. The fractal measurement used here seems to suggest that the inverse modelling from "fractal" pumping test data does indeed result in fracture networks with fractal characteristics. At this stage it is not valid to extrapolate from these results to the general case, further study would be necessary.

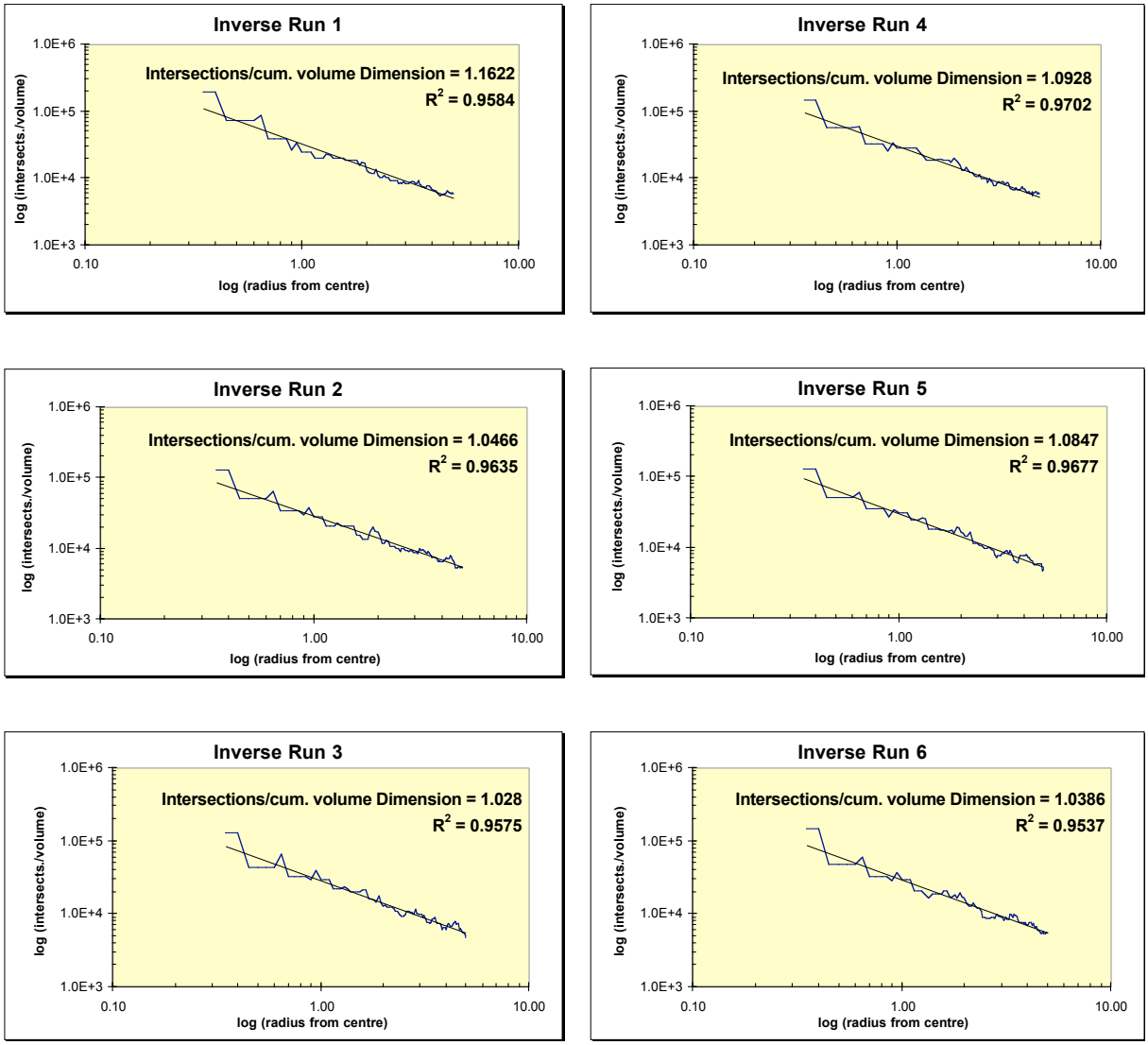


Figure 44: The fractal characteristics of the six networks obtained by inversion (c.f. figure 43).

5. Conclusions

A literature review has shown that fractal structures seem to be ubiquitous in geological materials and processes. Hence, a description of groundwater flow incorporating the principles of fractal scaling is probably not just a mathematical model but is based on the 'real' underlying structure of the aquifer. There are indications that the processes that control rock fracturing and faulting generate fractal fault structures. The non-integer flow dimension description of groundwater flow (Barker, 1988) has been compared with a description assuming an underlying "fractal" flow medium (Chang and Yortsos, 1990).
By comparing the two mathematical models and by simulating flow on synthetic networks it can be inferred that the fractal network parameter □, which is linked to network domain connectivity and tortuosity, is indeed an important system parameter. In theory, this parameter might be determined by a combination of hydraulic tests and fracture mapping. Further studies are necessary to evaluate the range and importance of this parameter from real pumping test data. This might be of significance for site investigations where a description of network topology is needed. Two examples are nuclear waste repository sites and geothermal extraction projects.

Computer code was developed and modified to generate synthetic 2-D fracture networks with the aim to allow the forward modelling of flow in fractal media. This code allows the generation of deterministic as well as stochastic fracture networks, allowing a large variety of synthetic fracture networks to be generated. It has been shown that it is possible to use 2-D synthetic fracture networks with a fractal geometric structure to simulate pumping tests with a flow behaviour as predicted by fractal flow theory, i.e. log-log plots of the pressure transient curve and its derivative with respect to logarithmic time fall on two parallel straight lines.

A functional relationship between fracturing probability, fractal geometric dimension and flow dimension has been revealed. Generally, a lower fractal mass dimension of the network also results in a lower flow dimension, as expected. However, this is not a one-to-one linear relationship, hence both are not equivalent as might be expected from mass scaling arguments.

Again, this is attributed to the parameter . Calibration curves have been determined for two different numbers of fracture generations that can be used to create fractal networks with a given flow dimension.

The methods developed for forward modelling were successfully used to simulate pumping tests in fractal media. The resulting pressure transient curves were then utilised to evaluate the feasibility of inverse modelling of fractal flow pressure transients. Using a slightly modified inverse modelling software developed at Lawrence Berkeley Lab. (Mauldon et al., 1993) that is based on a "simulated annealing" optimisation algorithm, it has been shown that inversion from fractal flow pressure transient data does indeed lead to fracture networks with fractal geometric properties, indicating that the analysis of pumping test data allows in principle a fractal characterisation of the underlying fracture network.

A new fractal measure, the radial count of fracture intersections divided by the cumulative fracture volume contained within a radial distance, was developed to characterise the fractal geometric properties of a fractal medium around a source zone (pumping well). If the inverse modelling of fractal flow from real data proved to be successful, this might be a method to characterise the underlying fracture network but more studies are needed to show the usefulness of the obtained parameter.

It was demonstrated that inverse modelling by "simulated annealing" allows to obtain a measure of uncertainty for the inverse run networks by generating several realisations of fractal networks that fit non-integer flow drawdown data. This can indicate where the amount of available data is sufficient to enable predictions of the actual network and shows where data are too sparse to allow any useful conclusions.

A lot of further studies using the developed computer code should and could be done in order to obtain a better understanding on how the use of fractal models can help to characterise flow in the sub-surface environment. The study presented here is only a first step towards a better understanding of fractal flow and it still needs to be demonstrated how fractal theory can be successfully applied to real hydraulic test data.

This study has also shown that there is a great potential in using more sophisticated methods in for the analysis of pumping test data and inverse modelling. It should be only a matter of

time until computer systems will be powerful enough for inverse modelling to be useful in standard investigations. To make proper use of this, however, theoretical studies need to be advanced further.

6. Recommendations

The theories and methods reported in this study should be applied to real pumping test data. A lot of further work could be done by combining the analysis of real pumping test data with other measurements, such as fracture mapping or geophysical data. This would allow the independent evaluation of fractal mass dimensions that can then be used to study how well fracture network connectivity and tortuosity are related to the fractal parameter θ and the percolation threshold. This, in turn, could be of great use for the characterisation of fracture network parameters in fields such as contaminant hydrogeology, nuclear waste repository site investigation and for work relating to the extraction of geothermal energy from fractured rocks.

A prime contender for the extension of this work towards a study on real data is the data set obtained by Nirex as part of the Sellafield rock characterisation programme, particularly the data from the long term RCF3 pumping test. This is because a very comprehensive data set exists that allows the independent validation of fracture network parameters and also provides the necessary hydrogeological background that is needed to set up appropriate models.
An initial evaluation suggests that the analysis of data from the Brockram formation might be suitable because flow-dimensions obtained from this rock formation show values that are consistently lower than two. Other data sets that might be useful include data from the Stripa-Mine project in Sweden and from the Yucca-Mountain project in the U.S.

The software developed as part of this project and the software supplied by Lawrence Berkeley Lab. should be used and enhanced for further studies. The fracture network generator could be modified to produce more realistic fracture networks. For example, it could be easily extended not only to produce networks with a fractal *geometric* structure but could also include a fractal scaling of fracture *apertures*. In this way, results from the inverse modelling step or from field data could be used to condition the generated networks, still retaining the stochastic nature of the algorithm. Before this is attempted, however, a better understanding of the effects of the current network parameters such as fracturing probability is necessary. As indicated, the analysis of data obtained from stochastic algorithms always requires several software runs to obtain a representative result. This should be done to correlate flow-

dimension and fractal dimension on synthetic fracture networks as presented in this study, but more different network parameters and sizes.

The fractal network generator as well as the inverse modelling code can be easily extended to a three-dimensional model. Instead of mapping the four corners of a square onto a new quadrilateral shape, the iterated functions system approach can be applied to a cube being mapped onto a six-sided fracture block, yielding 8 more coefficients for a set of equations similar to equation 32.

On the computer system used for this study, a UNIX machine with four 248MHz SPARC processors, the applied finite element flow simulation is still too slow to allow the analysis of real data in 3-D. However, improved algorithms, like a fast 3-D streamline routing technique as suggested by Vasco (1997) might solve this problem and allow the analysis of realistic three-dimensional networks in the near future.

The code is already fast enough to allow small 3-D inverse runs. One example how the current code might be used in three dimensions is to set up only a small and relatively sparse network, connecting the pumping well with the observation zones and inserting a few interconnecting bonds to create a sparse network. Inverse modelling on real data could then indicate how well different parts of the network are connected or an evaluation of which observation point is better connected to the pumping zone than another.

The code is flexible enough to conduct many further studies, the results of which are very important to further the understanding of fractal structures in hydrogeology.

7. Cited References

Abramowitz, M., Stegun, I.A.(eds.), 1964, Handbook of mathematical functions: with formulas, graphs, and mathematical tables, Nat. Bureau of Standards, New York

Acuna, J.A., Yortsos, Y.C., 1991, Numerical Construction and Flow Simulation on Networks of Fractures using Fractal Geometry, SPE Paper 22703, presented at the 66th Annual Conference and Exhibition of the Soc. of Petroleum Engineers, Dallas, Oct. 6.-9.

Acuna, J.A., Yortsos, Y.C. 1995a, Application of fractal geometry to the study of networks of fractures and their pressure transient, *Water Resources Research*, **31**, 3, 527-540

Acuna, J.A., Ershagi, I., Yortsos, Y.C., 1995b, Practical Application of Fractal Pressure-Transient Analysis in Naturally Fractured Reservoirs, *SPE Formation Evaluation*, **10**, 3, 173-179

Adler, P.M., 1985, Transport Processes in Fractals. 3. Taylor dispersion in 2 examples of fractal capillary networks, *International Journal of Multiphase Flow*, **11**, 2, 241-254

Akbarieh, M., Tawashi, R., 1989, Surface studies of calcium oxalate dihydrate single crystals during dissolution in the presence of stone-formers' urine, *Scanning Microscopy*, **3**, 1, 139-146

Alexander, S., Orbach, R., 1982, Density of states on fractals: <fractons>, *Le Journal de Physique - Lettres*, **43**, 17, 625-631

Armstrong, A.C., 1986, On the fractal dimensions of some transient soil properties, *Journal of Soil Science*, **37**, 641-652

Aviles, C.A., Scholz, C.H., Boatwright, J., 1987, Fractal analysis applied to characteristic segments of the San Andreas Fault, *Journal of Geophysical Research*, **92**, B1, 331-344

Avnir, D., Farin, D., Pfeifer, P., 1985, Surface geometric irregularity of particulate materials: the fractal approach, *Journal of Colloid and Interface Science*, **103**, 1, 112-123

Barker, J.A., 1988, A Generalized Radial Flow Model for Hydraulic Tests in Fractured Rock, *Water Resources Research*, **24**, 10, 1796-1804

Barnsley, M.F., 1988, Fractals everywhere, Boston Academic Press

Barton, C.C., Larsen, E., Fractal geometry of two-dimensional fracture networks at Yucca Mountain, southwestern Nevada, *Proc. Internat. Symp. Fundamentals Rock Joints*, 15-20 September, pp. 77-84

Beier, R.A., 1990, Pressure Transient Model of a Vertically Fractured Well in a Fractal Reservoir, paper SPE 20582 presented at the 65th Annual Technical Conference & Exhibition of the SPE, New Orleans, LA, Sept. 23-26

Black, J.H., Barker, J.A., Noy, D.J., 1986, Crosshole investigations: the method, theory and analysis of crosshole sinusoidal pressure tests in fissured rock, Stripa Proj., *Int. Rep. 86-03*, SKB, Stockholm

Black, J.H., 1994, Hydrogeology of fractured rocks - A question of uncertainty about geometry, *Applied Hydrogeology*, **3**, 56-70

Brown, W.K., Karpp, R.R., Grady, D.E., 1983, Fragmentation of the universe, *Astrophysics and Space Science*, **94**, 2, 401-412

Brown, S.R., Scholz, C.H., 1985, Broad bandwidth study of the topography of natural rock surfaces, *Journal of Geophysical Research*, **90**, B14, 12575-12582

Brown, S.R., 1987, A note on the description of surface roughness using fractal dimension, *Geophysical Research Letters*, **14**, 11, 1095-1098

Brown, S.R., 1989, Transport of Fluid and Electric Current through a Single Fracture, *Journal of Geophysical Research*, **94**, B7, 9429-9438

Burrough, P.A., 1983a, Multiscale sources of spatial variation in soil. I. The application of fractal concepts to nested levels of soil variation, *Journal of Soil Science*, **34**, 577-597

Burrough, P.A., 1983b, Multiscale sources of spatial variation in soil. II. A non-Brownian fractal model and its application in soil, *Journal of Soil Science*, **34**, 599-620

Carr, J.R., 1989, Fractal Characterization and Joint Surface Roughness in Welded Tuff at Yucca Mountain, Nevada, *Proc. 30th U.S. Symp. Rock Mechanics*, Morgantown, West Virginia, Ed., Khair, A.W., Balkema, Rotterdam, pp. 193-200

Chakrabarty, C., 1994, A note on fractional dimension analysis of constant rate interference tests, *Water Resources Research*, **30**, 7, 2339-2341

Chang, J., Yortsos, Y.C., 1990, Pressure-Transient Analysis of Fractal Reservoirs, *SPE Formation Evaluation*, **5**, 1, 31-38

Chang, J., Yortsos, Y.C., 1993, A Note on Pressure-Transient Analysis of Fractal Reservoirs, SPE Paper 25296, *SPE Advanced Technology Series*, **1**, 2, 170-171

Cox, B.L., Wang, J.S.Y., 1993, Fractal surfaces: Measurement and Applications in the Earth Sciences, *Fractals*, **1**, 1, 87-115

Curl, R.L., 1986, Fractal dimensions and geometrics of caves, *Mathematical Geology*, **18**, 8, 765-783

Datta-Gupta, A., Vasco, D.W., Long, J.C.S., dí Onfro, P.S., Rizer, W.D., 1995, Detailed Characterization of a fractured Limestone Formation by Use of Stochastic Inverse Approaches, *SPE Formation Evaluation*, **10**, 3, 133-140

Denley, D.R., 1990, Practical applications of scanning tunneling microscopy, *Ultramisc*., **33**, 83-92

Dershowitz, W., Redus, K., Wallmann, P., La Pointe, P., Axelsson, C-L., 1992, The implication of fractal dimension in hydrogeology and rock mechanics. Version 1.1, SKB Technical Report TR 92 - 17, 62 p.

Doe, T.W., 1991, Fractional dimension analysis of constant pressure well tests, paper SPE 22702 presented at the 66th Annual Technical Conference & Exhibition of the SPE, Dallas, Texas, Oct. 6-9

Dougherty, D.A., Marryott, R.A.,1991, Optimal groundwater management, 1, Simulated annealing, *Water Resources Research*, **27**, 10, 2493-2508

Edgar, G.A. (editor), 1993, Classics on fractals, Addison-Wesley Publ. Co.

*Follin, S., Thunvik, R., 1994, On the use of continuum approximations for regional modeling of groundwater flow through crystalline rocks, *Advances in Water Resources*, **17**, 133-145

Garrison, J.R., Pearn, W.C., von Rosenberg, D.U., 1992, The Fractal Menger Sponge and Sierpinski Carpet as models for reservoir rock/pore systems: I. Theory and Image Analysis of Sierpinski Carpets, *In Situ*, **16**, 4, 351-406

Garrison, J.R., Pearn, W.C., von Rosenberg, D.U., 1993a, The Fractal Menger Sponge and Sierpinski Carpet as models for reservoir rock/pore systems: II. Image analysis of Natural Fractal Reservoir Rocks, *In Situ*, **17**, 1, 1-53

Garrison, J.R., Pearn, W.C., von Rosenberg, D.U., 1993b, The Fractal Menger Sponge and Sierpinski Carpet as models for reservoir rock/pore systems: III. Stochastic simulation of Natural Fractal Processes, *In Situ*, **17**, 2, 143-182

Garrison, J.R., Pearn, W.C., von Rosenberg, D.U., 1993c, The Fractal Menger Sponge and Sierpinski Carpet as models for reservoir rock/pore systems: IV. Relationship of fractal dimension to the measured permeability of natural fractal reservoir rocks, *In Situ*, **17**, 3, 331-362

Hirata, T., 1989, Fractal dimension of fault systems in Japan, fractal structure in rock fracture geometry at various scales, *Pure and Applied Geophysics*, **131**, 1/2, 157-170

Huang, Z.H., Tian, J.F., Wang, Z.G., 1990, A study of the slit island analysis as a method for measuring fractal dimension of fractured surface*, Scripta Mettalur. et Mater*., **24**, 967-972

Isichenko, M.B., 1992, Percolation, statistical topography and transport in random media, *Rev. Modern Physics*, **64**, 4, 961-1043

Karasaki, K., 1986, Well Test Analysis in fractured Media, Ph.D.Thesis, Lawrence Berkeley Laboratory, University of California

Karasaki, K., Polek, J., Long, J., Witherspoon, P., 1988, Flow to wells in fractured reservoirs, Second Berkeley Symposium on Topics in petroleum Engineering, 9.-10. March, Earth Sciences Division, Lawrence Berkeley Laboratory, University of California

Kaye, B.H., 1986, The description of two-dimensional rugged boundaries in fine particle science by means of fractal dimension, *Powder Technology*, **46**, 245-254

Kaye, B.H., 1989, Image analysis techniques for characterizing fractal surfaces, in The Fractal Approach to Heterogeneous Chemistry, ed. Avnir, D., John Wiley & Sons, pp. 55-66

Katz, A.J., Thompson, A.H., 1985, Fractal sandstone pores, implications for conductivity and pore formation, *Phys. Rev. Lett*., **54**, 12, 1325

Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P., 1983, Optimization by simulated annealing, *Science*, **220**, 4598, 671-680

Krohn, C.E., Thompson, A.H., 1986, Fractal sandstone pores, automated measurements using scanning-electron-microscope images, *Phys. Rev*., **33**, B9, 6366-6374

Krohn, C.E., 1988a, Sandstone fractal and Euclidean pore volume distributions, *Journal of Geophysical Research*, **93**, B4, 3286-3296

Krohn, C.E., 1988b, Fractal measurements of sandstones, shales and carbonates, *Journal of Geophysical Research*, **93**, B4, 3297-3305

Laarhoven, P.J.M. van, Aarts, E.H.L., 1987, Simulated Annealing: Theory and Applications (Mathematics and its application), D. Reidel Publishing Co., Dordrecht, Holland

Langford, S.C., Zhenyi, M., Dickinson, J.T., 1989, Photo emission as a probe of chaotic processes accompanying fracture, *Journal of Material Sciences*, **4**, 5, 1272-1279

La Pointe, P.R., 1988, A method to characterize fracture density and connectivity through fractal geometry, *Int. J. Rock Mech. Min. Sci. & Geomech. Abstr*., **25**, 6, 421-429

La Pointe, P.R., 1994, Evaluation of stationary and non-stationary geostatistical models for inferring hydraulic conductivity values at Äspö, SKB Technical Report TR 94 - 22, 79 p.

Main, I.G., Meredith, P.G., Sammonds, P.R., Jones, C., 1990, Influence of fractal flaw distributions on rock deformation in the brittle field, in Deformation Mechanisms, Rheology and Tectonics, *Geol. Soc. Spec. Publ*., London, 54, 81-96

Mandelbrot, B., 1977, Fractals, W.H Freeman and Co., San Francisco

Mandelbrot, B.B., Passoja, D.E., Paullay, A.J., 1984, Fractal character of fracture surfaces of metals, *Nature*, **308**, 721-722

Mauldon, A.D., Karasaki, K., Martel, S.J., Long, J.C.S., Landsfeld, M., Mensch, A., 1993, An inverse technique for developing models for fluid-flow in fracture systems using simulated annealing, *Water Resources Research*, **29**, 11, 3775-3789

Mecholsky, J.J., Mackin, T.J., 1988, Fractal analysis of fracture in Ocala Chert*, Jour. Mater. Sci. Lett*., **7**, 1145-1147

Metropolis, N., et al., 1953, Equation of State Calculations by Fast Computing Machines, *J. of Chem. Physics*, **21**, 1087 - 1092

Miller, S.M., McWilliams, P.C., Kerkering, J.C., 1990, Ambiguities in estimating fractal dimensions of rock fracture surfaces, Rock Mechanics, Contributions and Challenges, eds. Hustrulid and Johnson, Balkema, Rotterdam

Najita, J.S., Karasaki, K., Peterson, J., 1994, Inversion of hydraulic well tests using cluster variable aperture simulated annealing, in: AGU 1994 fall meeting, Eos, Transactions, American Geophysical Union, **75**, 44, suppl., page 255.

Norton, D., Sorenson, S., 1989, Variations in geometric measure of topographic surfaces underlain by fractured granitic plutons, *Pure and Applied Geophysics*, **131**, 1, 77-97

Orbach, R., 1986, Dynamics of Fractal Networks, *Science*, **231**, 814-819

Odling, N.E., 1994, Natural Fracture Profiles, Fractal Dimension and Joint Roughness Coefficients, *Rock Mechanics and Rock Engineering*, **27**, 3, 135-153

Okuba, P.G., Aki, K., 1987, Fractal geometry in the San Andreas Fault System, *Journal of Geophysical Research*, **92**, B1, 345-355

O'Shaughnessy, B., Procaccia, I., 1985, Diffusion on Fractals, *Physical Review A*, **32**, 5, 3073-3083

Olsson, O.J., Black, J.G., Holmes, D., 1988, Site characterization and validation, stage 2 - Preliminary predictions, Rep. 88, Swed. Geol. Co., Stockholm

Pande, C.S., Smith, S., Richards, L.R., 1987, Fractal characteristics of fractal surfaces, *Journ. Mat. Sci. Lett*., **6**, 295-297

Press, W.H., et al., 1986, Numerical recipes : the art of scientific computing, Cambridge University Press

Robertson, M.C., Sammis, C.G., 1995, Fractal analysis of three-dimensional spatial distributions of earthquakes with a percolation iterpretation, *Journal of Geophysical Research*, **100**, B1, 609-620

Sahimi, M., Robertson, M.C., Sammis, C.G., 1993, Fractal distribution of earthquake hypocenters and its relation to fault patterns and percolation, *Phys. Rev. Lett*., **70**, 2186-2189

Sammis, C.G., King, G.C., Biegel, R., 1987, The kinematics of gouge deformation, *Pure Appl. Geophysics*, **125**, 777-812

Scholz, C.H., Aviles, C.A., 1986, The fractal geometry of faults and faulting, in Earthquake Source Mechanics, eds. Das, S., Boatwright, J., Scholz, C.H., Monogram 37, *Amer. Geophys. Union*, Washington D.C.

Scholz, C.H., Cowie, P.A., 1990, Determination of total strain from faulting using slip measurements, *Nature*, **346**, 837-839

Schlueter, E.M., Zimmermann, R.W., Cook, N.G.W., Witherspoon, P.A., 1991, Fractal dimensions of pores in sedimentary rocks and relationship to permeability, LBL report LBL-30583, University of Southern California

Snow, R.S., 1989, Fractal sinuosity of stream channels, *Pure and Applied Geophysics*, **131**, 1/2, 99-109

Sreenivasan, K.R., Prasad, R.R., Meneveau, C., Ramshankar, R., 1989, The fractal geometry of interfaces and the multifractal distribution of dissipation in fully turbulent flows, *Pure and Applied Geophysics*, **131**, 1/2, 43-60

Sun, Ne-Zheng, 1994, Inverse Problems in Groundwater modeling, Kluwer Academic Publishers, Dordrecht, Holland

Takayasu, H., 1989, Fractals in the physical sciences, Manchester University Press, Manchester

Thomas, A., Blin-Lacroix, J.L., 1989, Stochastic modeling of fractured reservoir, paper presented at *International Hot Dry Rock Geothermal Energy Conference*, Camborne School of Mines, Camborne, U.K.

Thompson, A.H., 1991, Fractals in rock physics, *Annual Rev. Earth Planet. Sci*., **19**, 237-262

Turcotte, D.L., 1986, Fractals and Fragmentation, *Journal of Geophysical Research*, **91**, B2, 1921-1926

Turcotte, D.L., 1992, Fractals and Chaos in Geology and Geophysics, Cambridge University Press

Underwood, E.E., Banerji, K., 1986, Fractals in Fractography, *Material Science and Engineering*, **80**, 1-14

Vasco, D.W., Datta-Gupta, A., 1997, Integrating multiphase production history in stochastic reservoir characterization, *SPE Formation Evaluation*, **12**, 3, 149-156

Watanabe, K., Takahashi, H., 1995, Fractal geometry characterization of geothermal reservoir fracure networks, *Journal of Geophysical Research*, **100**, B1, 521-528

8. Extended Bibliography

These references are not cited in the thesis text but are relevant to the subject field and allow further study by the interested reader.

Bangoy, L.M., Bidaux, P., Drogue, C., Plégat, R., Pistre, S., 1992, A new method of characterizing fissured media by pumping tests with observation wells, *Journal of Hydrology*, **138**, 77-88

Barker, J.A., 1985, Generalized well function evaluation for homogeneous and fissured aquifers, *Journal of Hydrology*, **76**, 143-154

Barker, J.A., 1991, The Reciprocity Principle and an analystical solution for Darcian flow in a network, *Water Resources Research*, **27**, 5 , 743-746

Braester, C., Zeitoun, D.G., 1992, Pressure transient response of stochastically heterogeneous fractured reservoirs, *Transport in Porous Media*, **11**, 263-280

Coakley, K.J., 1992, Spatial statistics for predicting fluid flow through a single rock fracture, *Mathematical Geology*, **24**, 8, 905-927

da Prat, G., 1990, Well test analysis for fractured reservoir evaluation, Elsevier, Amsterdam

Gavrilenko, P., Guéguen, Y., 1998, Flow in fractured media: A modified renormalization method, *Water Resources Research*, **34**, 2, 177-191

Harris, C.K., 1992, Effective-medium treatment of flow through anisotropic fracture system - Improved permeability estimates using a new lattice mapping, *Transport in Porous Media*, **9**, 287-295

Hull, L.C., Koslow, K.N., 1986, Streamline routing through fracture junctions, *Water Resources Research*, **22**, 12, 1731-1734

Kolditz, O., 1995, Modelling flow and heat transfer in fractured rocks: Conceptual model of a 3-D deterministic fracture network, *Geothermics*, **24**, 3, 451-470

Long, J.C.S., Billaux, D.M., 1987, From field data to fracture network modeling: An example incorporating spatial structure, *Water Resources Research*, **23**, 7, 1201-1216

Mohrlok, U., Liedl, R., 1996, Generating fracture networks using iterated function systems, *Geologische Rundschau*, **85**, 92-95

Narasimhan, T.N., 1985, Geometry-Imbedded Darcy's law and Transient Subsurface Flow, *Water Resources Research*, **21**, 8, 1285-1292

Appendix A - Excel functions to compute fractal pressure-transient data

This appendix shows the Visual Basic code that was used within EXCEL to generate data points of fractal pressure transients and their derivative with respect to logarithmic time as a function of dimensionless radius, dimensionless time, delta and theta. The code includes functions that compute the gamma and incomplete gamma functions $\Gamma(x)$ and $\Gamma(a,x)$ as defined in the main text. Relationships used to base these on other functions can be found in Press et al. (1986) and Abramowitz and Stegun (1964). These functions can also be used to compute the functions given by Barker (1988). Most functions were translated into Visual Basic directly from Press et al. (1986). Comments are indicated by the character ‘.

The following relationships were used (Abramowitz and Stegun, 1964):

1. $\Gamma(p,x) = \Gamma(p)[1 - P(p,x)]$ for p>0, since

   $P(p+1,x) = P(p,x) - \frac{x^p e^{-x}}{\Gamma(p+1)}$ and $\Gamma(p+1) = p\Gamma(p)$

2. $\Gamma(p,x) = \frac{\Gamma(p+1)}{p}\left[1 - P(p+1,x)\right] + \frac{x^p e^{-x}}{\Gamma(p+1)} = \frac{\Gamma(p+1)[1 - P(p+1,x)] - x^p e^{-x}}{p}$

3. $\Gamma(0,x) = E_1(x)$

```
Option Explicit 'to facilitate debugging (forces declarations)
Const pi As Double = 3.14159265358979

Function fractal_pressure(r As Double, t As Double, delta As Double, theta
As Double) As Double
' This function computes the fractal pressure drawdown as shown in
  equation (24)

  Dim p As Double
  p = r ^ ((2 + theta) * (1 - delta))
  p = p / (gamma(delta) * (2 + theta))
  p = p * gammi((delta-1), (r ^ (2 + theta) / ((2 + theta)^2 *t)))
  fractal_pressure = p
End Function

Function fractal_pressure_der(r As Double, t As Double, delta As Double,
theta As Double) As Double
' This function computes the fractal drawdown derivative with respect to
logarithmic time as shown in equation (24)

  Dim p As Double
  Dim y As Double
  y = r ^ (theta + 2) / ((theta + 2) ^ 2 * t)
  p = Exp(-y) * t ^ (1 - delta) / (gamma(delta) * (theta + 2) ^ (2
    * delta - 1))
  fractal_pressure_der = p
End Function

Function gamma(ByVal p As Double) As Double
' The standard gamma function
  Dim z As Double
  If p < 1 Then         ' if argument <1, use reflection formula
                  iteratively
    z = 1 - p
    gamma = pi * z / Sin(pi * z) / gamma(z + 1)
  Else
    gamma = Exp(gammln(p))    'the gamma function is easier computed
                         in logarithmic form, hence this
End If
End Function



Function gammln(ByVal xx As Double) As Double
' this gives the natural log. of the gamma function
Const stp As Double = 2.50662827465
Const HALF As Double = 0.5
Const one As Double = 1#
Const fpf As Double = 5.5

Dim X As Double, tmp As Double, ser As Double
Dim j As Integer


Const cof1 As Double = 76.18009173
Const cof2 As Double = -86.50532033
Const cof3 As Double = 24.01409822
Const cof4 As Double = -1.231739516
Const cof5 As Double = 0.00120858003
Const cof6 As Double = -0.00000536382
   X = xx - one
   tmp = X + fpf
   tmp = (X + HALF) * Log(tmp) - tmp
```

```
   ser = one
   'FOR j := 1 TO 6 DO BEGIN  'this loop normally initialises
   '   x := x+one;                    'the constants for the series
                                      'computation, but easier this way
   '   ser := ser+cof[j]/x
   'END;
   X = X + one
   ser = ser + cof1 / X
   X = X + one
   ser = ser + cof2 / X
   X = X + one
   ser = ser + cof3 / X
   X = X + one
   ser = ser + cof4 / X
   X = X + one
   ser = ser + cof5 / X
   X = X + one
   ser = ser + cof6 / X

   gammln = tmp + Log(stp * ser)
End Function


Function gammp(a As Double, X As Double) As Double
Dim gammcf As Double, gln As Double

   If ((X < 0#)) Then ' Or (a <= 0#)) Then
      Error (65532) 'corresponds to error below
      'writeln('pause in GAMMP - invalid arguments'); readln
   End If
   If (X < (a + 1#)) Then
      Call gser(a, X, gammcf, gln)
      gammp = gammcf
   Else
      Call gcf(a, X, gammcf, gln)
      gammp = 1# - gammcf
   End If
End Function

Sub gser(a As Double, X As Double, gamser As Double, gln As Double)

Const itmax As Integer = 100
Const EPS As Double = 0.0000003
Dim n As Integer
Dim sum As Double, del As Double, ap As Double

gln = gammln((a))
   If (X <= 0#) Then          'depending on the paramter, different
                              'methods of computation are chosen
      If (X < 0#) Then
         Error (65531) corresponds to error below
      'writeln('pause in GSER - x less than 0'); readln
      End If
      gamser = 0#
   Else
      ap = a
      sum = 1# / a
      del = sum
      For n = 1 To itmax
         ap = ap + 1#
         del = del * X / ap
         sum = sum + del
         If (Abs(del) < Abs(sum) * EPS) Then GoTo 1
```

```
      Next
      Error (65530) corresponds to error below
'writeln('pause in GSER - a too large, itmax too small'); readln;
1:
      gamser = sum * Exp(-X + a * Log(X) - gln)
   End If
End Sub


Sub gcf(a As Double, X As Double, gammcf As Double, gln As Double)
Const itmax As Integer = 100
Const EPS As Double = 0.0000003
Dim n As Integer
Dim gold As Double, g As Double, fac As Double, b1 As Double
Dim b0 As Double, anf As Double, ana As Double, an As Double
Dim a1 As Double, a0 As Double

   gln = gammln((a))
   gold = 0#
   a0 = 1#
   a1 = X
   b0 = 0#
   b1 = 1#
   fac = 1#
   For n = 1 To itmax
      an = 1# * n
      ana = an - a
      a0 = (a1 + a0 * ana) * fac
      b0 = (b1 + b0 * ana) * fac
      anf = an * fac
      a1 = X * a0 + anf * a1
      b1 = X * b0 + anf * b1
      If (a1 <> 0#) Then
         fac = 1# / a1
         g = b1 * fac
         If (Abs((g - gold) / g) < EPS) Then GoTo 2
         gold = g
      End If
   Next
      Error (65535) ' corresponds to error below
      'writeln('pause in GCF - a too large, itmax too small');
      'readln;
2:   gammcf = Exp(-X + a * Log(X) - gln) * g
End Sub

Function gammi(p As Double, X As Double) As Double
' incomplete gamma functio for p > -1
' note that this gives the incomplete gamma function for p>-1!
' most other implementations are restricted to positive arguments!
' gamdis is the cumulative gamma distribution function (see below)
Const EPS As Double = 0.01
If (p > EPS) Then
  gammi = gamma(p) * (1# - gamdis(X, p))
ElseIf (p < (-EPS)) Then
  gammi = (gamma(1# + p) * (1# - gamdis(X, 1# + p)) - Exp(-X) * X ^ p) / p
Else
  gammi = expint(1, X)
End If
End Function

Function expint(n As Integer, X As Double) As Double
'Evaluates the exponential integral En(x).
Const MAXIT As Integer = 100 'Maximum allowed number of iterations.
```

```
Const EULER As Double = 0.5772156649 'Euler's constant .
Const FPMIN As Double = 1E-30 'Close to smallest representable floating-
point number.
Const EPS As Double = 0.000000001 'Desired relative error, not smaller than
the machine precision.
'void nrerror(char error_text[]);
Dim i As Integer
Dim ii As Integer
Dim nm1 As Integer
Dim a As Double
Dim b As Double
Dim c As Double
Dim d As Double
Dim e As Double
Dim del As Double
Dim fact As Double
Dim h As Double
Dim psi As Double
Dim ans As Double

nm1 = n - 1
If ((n < 0) Or (X < 0#) Or ((X = 0#) And (n = 0 Or n = 1))) Then
  Error (65535)
  Exit Function
  'nrerror("bad arguments in expint")
Else
  If (n = 0) Then
    ans = Exp(-X) / X ' Special case.
  Else
    If (X = 0#) Then
      ans = 1# / nm1 ' Another special case.
    Else
      If (X > 1#) Then
' { Lentz's algorithm ( x 5.2).
        b = X + n
        c = 1# / FPMIN
        d = 1# / b
        h = d
        For i = 1 To MAXIT
          a = -i * (nm1 + i)
          b = b + 2#
          d = 1# / (a * d + b) ' Denominators cannot be zero.
          c = b + a / c
          del = c * d
          h = h * del
          If (Abs(del - 1#) < EPS) Then
            ans = h * Exp(-X)
            expint = ans
            Exit Function
          End If
        Next
        Error (65532) ‘ corresponds to error below
      'nrerror("continued fraction failed in expint");
        Exit Function
      Else  ' Evaluate series.
        If (nm1 <> 0) Then 'Set first term.
          ans = 1 / nm1
        Else
          ans = -Log(X) - EULER
        End If
        fact = 1#
        For i = 1 To MAXIT
          fact = fact * (-X / i)
```

```
          If (i <> nm1) Then
            del = -fact / (i - nm1)
          Else
            psi = -EULER   ' Compute (n).
            For ii = 1 To nm1
              psi = psi + 1# / ii
            Next
            del = fact * (-Log(X) + psi)
          End If
          ans = ans + del
          If (Abs(del) < Abs(ans) * EPS) Then
            expint = ans
            Exit Function
          End If
        Next
        Error (65531) ' nrerror("series failed in expint");
        Exit Function
      End If
    End If
  End If
End If
expint = ans
End Function



Function gamdis(X As Double, a As Double) As Double

'C     Calculates the gamma distribution function
'C
'C        G(x,a) = (1/gamma(a)) * int(0,x)[exp(-t) * t**(a-1)] dt.
'C
'C     Based on
'C        W. Gautschi, ALGORITHM 542 Incomplete Gamma Functions,
'C        ACM Trans. Math. Software 5 (1979) 482-489.'

Const EPS As Double = 0.00001
Const EPS1 As Double = 0.0000005
Const ALH As Double = -0.6931472
Const Z1 As Double = 1
Const HALF As Double = Z1 / 2
Const QUAR As Double = Z1 / 4
Const C1 As Double = 3 / 2 * Z1
Const KMAX As Integer = 300

Dim hst As Double
Dim alx As Double
Dim alfa As Double
Dim k As Integer
Dim p As Double
Dim q As Double
Dim r As Double
Dim s As Double
Dim t As Double
Dim u As Double
Dim y As Double
Dim rho As Double
Dim ga As Double
Dim term As Double
Dim sum As Double

Static c(14) As Double
```

```
c(1) = 0.5772157
c(2) = -0.6558781
c(3) = -0.0420026
c(4) = 0.1665386
c(5) = -0.0421977
c(6) = -0.009622
c(7) = 0.0072189
c(8) = -0.0011652
c(9) = -0.0002152
c(10) = 0.0001281
c(11) = -0.0000201
c(12) = -0.0000013
c(13) = 0.0000011
c(14) = -0.0000002

      hst = 0
      If (X = 0) Then
        gamdis = hst
        Exit Function
      End If
      If ((X < 0) Or (a <= 0)) Then
       'WRITE(ERRTXT,101) X,A
       'CALL MTLPRT(NAME,'G106.1',ERRTXT)
       'ILLEGAL ARGUMENT(S) X = ',E15.8,'  A = ',E15.8)
       Error (65535)
       Exit Function
      Else
       alx = Log(X)
      End If
      If (X < QUAR) Then
       alfa = ALH / alx
      Else
       alfa = X + QUAR
      End If
      If (a > alfa) Then
       term = 1
       sum = 1
       For k = 1 To KMAX
         term = X * term / (a + k)
         sum = sum + term
         If (Abs(term) <= EPS * sum) Then Exit For
       Next
       If k = KMAX Then
         Error (65535) 'no convergence
         Exit Function
       End If
       hst = sum * Exp(a * alx - X - gammln(1 + a))
      ElseIf (X > C1) Then
       p = 0
       s = 1 - a
       q = (X + s) * (X - 1 - a)
       r = 4 * (X + s)
       term = 1
       sum = 1
       rho = 0
       For k = 2 To KMAX
         p = p + s
         q = q + r
         r = r + 8
         s = s + 2
         t = p * (1 + rho)
         rho = t / (q - t)
         term = rho * term
```

```
          sum = sum + term
          If (Abs(term) <= EPS * sum) Then Exit For
        Next
        If k = KMAX Then
          Error (65533) 'no convergence
          Exit Function
        End If
        hst = 1 - (a * sum / (X + 1 - a)) * Exp(a * alx - X - gammln(1 + a))
      Else
        If (a < HALF) Then
         sum = c(14)
         For k = 13 To 1 Step -1
           sum = a * sum + c(k)
         Next
         ga = -sum / (1 + a * sum)
         y = a * alx
         If (Abs(y) >= 1) Then
          u = ga - (Exp(y) - 1) / a
         Else
          sum = 1
          term = 1
          For k = 2 To KMAX
            term = y * term / k
            sum = sum + term
            If (Abs(term) <= EPS1 * sum) Then Exit For
          Next
        If k = KMAX Then
          Error (65532) 'no convergence
          Exit Function
        End If
         u = ga - sum * alx
         End If
        Else
         u = gamma(a) - Exp(a * alx) / a
        End If
        p = a * X
        q = a + 1
        r = a + 3
        term = 1
        sum = 1
        For k = 2 To KMAX
          p = p + X
          q = q + r
          r = r + 2
          term = -p * term / q
          sum = sum + term
          If (Abs(term) <= EPS1 * sum) Then Exit For
        Next
        If k = KMAX Then
          Error (65531) 'no convergence
          Exit Function
        End If
        hst = 1 - a * (u + sum * Exp((1 + a) * alx) / (1 + a)) / gamma(1 +
a)
      End If
      gamdis = hst
End Function
```

Appendix B - Code in C to generate modified Sierpinski gasket networks

This appendix lists the code that was used to generate the modified Sierpinski network. It is partly based on code supplied by Jorge Acuna (Unocal), but largely rewritten, extended and debugged. It is written in portable ANSI-C but using C++ style comments (//). The code, as shown below, compiles on Sun Solaris without change with the standard cc compiler system. Instructions for the right choice of compiling parameters are given in the electronic supplement. Three lines of code need to be uncommented to compile this code on the Metroworks Codewarrior compiler for Macintosh and are indicated in the code (two include lines at the beginning and one argument parsing command right at the beginning of the main). This code only needs to be slightly modified to create stochastic fracture networks. Since it is largely the same, it is only included on disk. Annotated input and output files are also included on disk.
Additional code is needed to postprocess the output files to make the network well-behaved for input into the inverse modelling code. This is shown in appendix C.

```
// #include <console.h>       // use these two lines when compiling
// #include <SIOUX.h>         // with Macintosh Codewarrior

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>
#define MAXGENS 16            // the maximum number of fracture generations
#define MAXNODES ((1<<MAXGENS)+6) // +6 in order to have four corner nodes

                              // Note: at the moment the memory allocation
                              // is determined by these two constants
                              // it would be much better to introduce
                              // dynamic memory allocation with malloc
/* Global variables */
static double xcoord[6+1],ycoord[6+1];// xcoord[1..4], ycoord[1..4] are
                                      // coordinates of original box
                                      // xcoord[5..6], ycoord[5..6] are
                                      // coordinates of first fracture
static double coeffs[10+1][10+1];     // coefficients for bilinear
                                      // coordinate transformation
static long ivec[20+1];
static int mafrac[MAXNODES/2+4];      // this array stores information
                                      // about which fractures are includes
static double nodes[MAXNODES][4+1];
static int node_isc[MAXNODES];        // records position of nodes
                                      // 0=internal; 1= bottom;
                                      // 2=left;3=top;4=right
                                      // corners are always top or bottom
static int node_ib[MAXNODES];         // records boundary type: 1 = const
                                      // head; 2 = flux(check)
static int elements[MAXNODES*2][2+1];
static int node_conversion[MAXNODES];

static int topnodes[MAXNODES/4];      // used to generate boundary elements
static int bottomnodes[MAXNODES/4];
static int leftnodes[MAXNODES/4];
static int rightnodes[MAXNODES/4];
int nleft=1, nright=1, ntop=1, nbottom=1;
// counts number of nodes along edges

static double pfrac;                          // fracture probability
static int iupctof;                           // upper cut off length
scale (random fracturing starts from here)
static int ilwctof;                           // lower cut off length
scale. This is the last generation of fractures
static int iseed;                             // seed for random number
generator
static int nelmts;                                  // number of elements
(excluding fractures that follow an unfractured generation
static int elmtcount;                         // counts how many element
data are written
static int nodecount;
static int nnodes;                                  // number of nodes
static double trans, aperture, stor;   // transmissivity, aperture, storage
static double head;                                 // orig. boundary
value (head)
static int bound_type;                        // boundary type of edge
nodes: 0: none, 1: head fixed, 2: flow fixed
static double rot;      // rotate nodes by rot degrees during output
                        // not implemented yet, but only needs sin/cos
                        // in output loop
static FILE *nodefile, *elmtfile, *xyzfile, *nodexyzfile;
```

```
/* */

/* Prototypes for functions */
void read_parameters(char *);
void calculate_coeffs(void);
void transform(int ,double ,double ,double * ,double *);
void fracture(int , int );
void fractal(double , int , int );
void connectivity(int, int, int);
void output();
void generate_border();
static int comparex(const int *, const int *);
static int comparey(const int *, const int *);

/* */

main(int argc, char *argv[])
{
  int ifrac; // ifrac is number of fracture for a given generation, i.e. 1,
             // 1-2, 1-4 etc.
  int i,j,k; // loop counters
  long ires;

 //  argc = ccommand(&argv);   // use this for Macintosh

  if (argc<2)
  {
    printf("This program expects one command line argument specifying the
parameter file.\n");
    exit(1);
  }

  if ( (nodefile = fopen("fracnode.inp","w")) ==0)
  {
    printf("Can't open node output file for writing. Exit.\n");
    exit(1);
  }
  else
    fprintf(stdout, "opened fracnode for writing\n");
  if ( (elmtfile = fopen("fracelmt.inp","w")) ==0)
  {
    fclose(nodefile);
    printf("Can't open node output file for writing. Exit.\n");
    exit(1);
  }
  else
    fprintf(stdout, "opened fracelmt for writing\n");
  if ( (xyzfile = fopen("fracxyz.inp","w")) ==0)
  {
    fclose(nodefile);
    fclose(elmtfile);
    printf("Can't open xyz output file for writing. Exit.\n");
    exit(1);
  }
  else
    fprintf(stdout, "opened fracxyz for writing\n");
  if ( (nodexyzfile = fopen("nodexyz.dat","w")) ==0)
  {
    fclose(nodefile);
    fclose(elmtfile);
    fclose(xyzfile);
    printf("Can't open nodexyz output file for writing. Exit.\n");
```

```
    exit(1);
  }
  else
    fprintf(stdout, "opened nodexyz.dat for writing\n");

  fprintf(stdout, "calling read_parameters with argument: %s\n", argv[1]);

  read_parameters(argv[1]);
  srand(iseed);                     // initializes random seed
  calculate_coeffs();               // determines bi-linear coefficients
                                    // from co-ordinates of initiator
                                    // and initial fracture
  fprintf(stdout, "calculated coefficients.\n");
  fractal(pfrac, iupctof, ilwctof);
      // this routine does all the fracture transformations
      // pfrac is the fracturing probability, iupctof&ilwctof are
      // upper and lower cut-off scale
  fprintf(stdout, "finished with fractal.\n");

  ifrac = 1;
  for (i=1; i<= ilwctof; I++) // go through all fracture generations
  {
    for(j=1; j<=(1<<i); j++)
    {
      ifrac++;                // fracture segment counter for each gen.
      if (mafrac[ifrac] == 1)
      {
        ires = j - 1;
        for (k=1; k<= i; k++)
        {
          ivec[k] = ires / (1<<(i-k)) +1;   // a simple binary count
          ires -= (ivec[k]-1) * (1<<(i-k)); // done by permuation
        }
        fracture(ifrac, I);   // this determines whether a given fracture
      }                       // is further subdivides (defines Sierpinski)
    }
  }
  fprintf(stdout, "finished with fractures.\n");
  elmtcount = 1;  // initialize element count for connectivity
  nodecount = 1;
  connectivity(1, 2, ilwctof);      // generates additional nodes where
  for (i=2; i<= ((1<<(ilwctof+1))-1); i++) // fractures intersect
    if (mafrac[i] ==1) connectivity((2*i+1), (2*i+2), ilwctof);
  //elmtcount--; // reset elmtcount(one too many in connectivity)
  //nodecount--;
  for (i=0; i<=3; i++)
  {
    nodes[(MAXNODES-4+i)][1]=xcoord[i+1];
    nodes[(MAXNODES-4+i)][2]=ycoord[i+1];
    node_conversion[(MAXNODES-4+i)]=nodecount++;
  }
  nodecount--;
  //elmtcount--; // remove this if running with generate_border()
  generate_border();
  output();
  fclose(elmtfile);
  fclose(nodefile);
  fclose(xyzfile);
  fclose(nodexyzfile);
  fprintf(stdout, "execution complete. exiting.\n");

  return(0);
}
```

```
void read_parameters(char *filename) // this reads in the parameter file
{                                    // see that file for description
  FILE *fp;                          // of parameters
  int i;

  if ( (fp=fopen(filename,"r")) ==0)
  {
    printf("Error: can't open parameter file %s. Exit.\n", *filename);
    exit(1);
  }

  for (i=1;i<=6;i++)
  {
    if (fscanf(fp, "%lf%lf", &(xcoord[i]), &(ycoord[i])) <2)
      {
        printf("error reading in coordinates from parameter file at step
%d. Exit\n", i);
        exit(1);
      }
    printf(" xcoord[%d]: %5f  ycoord[%d]:  %5f\n", i, xcoord[i], i,
ycoord[i]);
  }
    (void) fscanf(fp, "%lf%d%d", &pfrac, &iupctof, &ilwctof);
    (void) fscanf(fp, "%d", &iseed);
    (void) fscanf(fp, "%lf%lf%lf", &trans, &aperture, &stor);
    (void) fscanf(fp, "%lf", &head);
    (void) fscanf(fp, "%d", &bound_type);
    (void) fscanf(fp, "%lf", &rot);

    fprintf(stdout, "pfrac %f iupctof %d ilwctof %d \n", pfrac, iupctof,
ilwctof);
    fprintf(stdout, "iseed %d\n", iseed);
    fprintf(stdout, "T %le Apert %le S %le\n", trans, aperture, stor);
    fprintf(stdout, "head: %e\n", head);
    fprintf(stdout, "bound_type: %d\n", bound_type);
    fprintf(stdout, "rotate by (degrees): %le\n", rot);

    fprintf(stdout, "leaving read_parameters\n");
  fclose(fp);
}


void calculate_coeffs()
/* this function solves the bi-linear equations that are used to map co-
ordinates in order to determine the 8 coefficients needed. It is only
called once at the beginning of program execution. */
{
  int i;                // loop counters
  double P21, P31, P41;
  double q,r,s;
  double num, den;
  int n, m;
  double X[10][10];
  double Y[10][10];

  nodes[1][1] = xcoord[5];
  nodes[1][2] = ycoord[5];
  nodes[1][3] = 0.0;
  nodes[1][4] = 256.0;  //check this!!
  nodes[2][1] = xcoord[6];
  nodes[2][2] = ycoord[6];
```

```
  nodes[2][3] = 512.0;
  nodes[2][4] = 256.0;  //check this!!
  nodes[3][3] = -1.0;
  nodes[3][4] = -1.0;
  nodes[4][3] = -1.0;
  nodes[4][4] = -1.0;
  nodes[3][1] = -10.0;
  nodes[3][2] = -10.0;
  nodes[4][1] = -10.0;
  nodes[4][2] = -10.0;

  X[1][1] = xcoord[5]; // not very neat, but it works.
  Y[1][1] = ycoord[5];
  X[1][2] = xcoord[1];
  Y[1][2] = ycoord[1];
  X[1][3] = xcoord[2];
  Y[1][3] = ycoord[2];
  X[1][4] = xcoord[6];
  Y[1][4] = ycoord[6];
  X[2][1] = xcoord[4];
  Y[2][1] = ycoord[4];
  X[2][2] = xcoord[5];
  Y[2][2] = ycoord[5];
  X[2][3] = xcoord[6];
  Y[2][3] = ycoord[6];
  X[2][4] = xcoord[3];
  Y[2][4] = ycoord[3];

  for (i=1; i<=2; i++)
  {
    P21 = xcoord[2] * ycoord[2] - xcoord[1] * ycoord[1];
    P31 = xcoord[3] * ycoord[3] - xcoord[1] * ycoord[1];
    P41 = xcoord[4] * ycoord[4] - xcoord[1] * ycoord[1];

    q = (X[i][3] - X[i][1]) * P21 - (X[i][2] - X[i][1]) * P31;
    r = (xcoord[3] - xcoord[1]) * P21 - (xcoord[2] - xcoord[1]) * P31;
    s = (ycoord[3] - ycoord[1]) * P21 - (ycoord[2] - ycoord[1]) * P31;
    num = P21 * (X[i][4] - X[i][1] - q / s * (ycoord[4] - ycoord[1])) - P41
* (X[i][2] - X[i][1] - q / s * (ycoord[2] - ycoord[1]));
    den = P21 * (xcoord[4] - xcoord[1] - r / s * (ycoord[4] - ycoord[1])) -
P41 * (xcoord[2] - xcoord[1] - r / s * (ycoord[2] - ycoord[1]));
    n = 2 * i - 1;
    m = 2 * i;
    coeffs[n][1] = num / den;
    coeffs[n][2] = (q - r * coeffs[n][1]) / s;
    coeffs[n][3] = (X[i][2] - X[i][1] - coeffs[n][1] * (xcoord[2] -
xcoord[1]) - coeffs[n][2] * (ycoord[2] - ycoord[1])) / P21;
    coeffs[n][4] = X[i][1] - coeffs[n][1] * xcoord[1] - coeffs[n][2] *
ycoord[1] - coeffs[n][3] * xcoord[1] * ycoord[1];

    q = (Y[i][3] - Y[i][1]) * P21 - (Y[i][2] - Y[i][1]) * P31;
    num = P21 * (Y[i][4] - Y[i][1] - q / s * (ycoord[4] - ycoord[1])) - P41
* (Y[i][2] - Y[i][1] - q / s * (ycoord[2] - ycoord[1]));
    den = P21 * (xcoord[4] - xcoord[1] - r / s * (ycoord[4] - ycoord[1])) -
P41 * (xcoord[2] - xcoord[1] - r / s * (ycoord[2] - ycoord[1]));
    coeffs[m][1] = num / den;
    coeffs[m][2] = (q - r * coeffs[m][1]) / s;
    coeffs[m][3] = (Y[i][2] - Y[i][1] - coeffs[m][1] * (xcoord[2] -
xcoord[1]) - coeffs[m][2] * (ycoord[2] - ycoord[1])) / P21;
    coeffs[m][4] = Y[i][1] - coeffs[m][1] * xcoord[1] - coeffs[m][2] *
ycoord[1] - coeffs[m][3] * xcoord[1] * ycoord[1];
  }
}
```

```
void transform(int n, double xx, double yy, double *rxx, double *ryy)
/* this routine does the actual co-ordinate transform using the
coefficients computed by calculate coeffs() */
{
  int nn,mm;

  nn = 2*n -1;
  mm = 2*n;
  *rxx = coeffs[nn][1] * xx + coeffs[nn][2] * yy + coeffs[nn][3] * xx * yy
+ coeffs[nn][4];
  *ryy = coeffs[mm][1] * xx + coeffs[mm][2] * yy + coeffs[mm][3] * xx * yy
+ coeffs[mm][4];
}

void fracture(int ifrac, int n)
{

  double xx,yy, xref, yref, rxx, ryy, xxref, yyref;
  int ii,jj,kk;                      // loop counter

  for (ii=1; ii<=2; ii++)
  {
    xx   = nodes[ii][1];
    yy   = nodes[ii][2];
    xref = nodes[ii][3];
    yref = nodes[ii][4];
    for (jj=1; jj<=n; jj++)
    {
      kk = n - jj + 1;
      xxref = yref;
      yyref = -0.5 * xref + (256 + 256 * (ivec[kk] - 1)); // check this!
      transform(ivec[kk], xx, yy, &rxx, &ryy);
      xx = rxx;
      yy = ryy;
      xref = xxref;
      yref = yyref;
    }
    nodes[(2 * ifrac + ii)][1] =  xx;
    nodes[(2 * ifrac + ii)][2] =  yy;
    nodes[(2 * ifrac + ii)][3] =  (xref);
    nodes[(2 * ifrac + ii)][4] =  (yref);
  }
}

void fractal(double pfrac, int iupctof, int ilwctof)
{
  int i,j;  // loop counters
  static int vector[MAXNODES/2];
  int icnt, rnelim, ielim;
  long random;
  int flag;

  mafrac[1] = 1;
  flag=-1;
  for (i=1; i <= (iupctof -1); i++)
    for (j=(1<<i); j<=((1<<(i+1))-1); j++)
    {
      mafrac[j] = 1;
    }
  for (i=iupctof; i<= ilwctof; i++)
  {
    icnt = 1;
```

```
    for (j=(1<<(i-1)); j <= ((1<<i) - 1); j++)
    {
      if (mafrac[j]==1)
      {
        icnt += 2;                  // this loop here implements
        vector[icnt-1] = 2*j;       // the generation of the Sierpinski
        vector[icnt]   = 2*j + 1;
        if ((i %2)==0)              // for every second fracture generation
          if (((icnt+flag)%4)==0)
            vector[icnt+(-flag-1)/2]=-1; // need to take into account
                                         // rotation as well, in order
                                         // to always leave out the
                                         // top-right corner
      }
    }
    if ((i%2)==0) flag *= -1;

    for (j=2; j<= icnt; j++)
      if (vector[j] > 0) mafrac[vector[j]] = 1;
  }
}

void connectivity(int n1, int n2, int ilwctof)
/* This routine determines which nodes are used for the final network,
depending on fracture intersections */
{
 static int vector[MAXNODES];
  int min, max;
  int icnt;
  int iprov;
  int i,j;
  double dist, distmin;

  if (elmtcount >= MAXNODES)
  {
    printf("ERROR: reached maximum of allowed nodes (MAXNODES).Exit");
    exit(1);
  }
  vector[1] = n1;
  vector[2] = n2;
  if (nodes[n1][3] == nodes[n2][3])
  {
    if (nodes[n1][4] > nodes[n2][4])
    {
      min = nodes[n2][4];
      max = nodes[n1][4];
    }
    else
    {
      min = nodes[n1][4];
      max = nodes[n2][4];
    }
    icnt = 2;
    for (i=1; i<=((1<<(ilwctof+2))); i++)
    {
      if ((nodes[i][3] == nodes[n1][3]) && (nodes[i][4] > min) &&
(nodes[i][4] < max))
      {
        icnt++;
        vector[icnt] = i;
      }
    }
  }
```

```
  if ((nodes[n1][4] == nodes[n2][4]))
  {
    if (nodes[n1][3] > nodes[n2][3])
    {
      min = nodes[n2][3];
      max = nodes[n1][3];
    }
    else
    {
      min = nodes[n1][3];
      max = nodes[n2][3];
    }
    icnt = 2;
    for (i=1; i<=((1<<(ilwctof+2))); i++)
    {
      if ((nodes[i][4] == nodes[n1][4]) && (nodes[i][3] > min) &&
(nodes[i][3] < max))
      {
        icnt++;
        vector[icnt] = i;
      }
    }
  }
  for (i=2; i<= icnt; i++)
/* this loop makes sure that only nodes of the same or smaller generation
are connected and finally outputs the results */
  {
    distmin = 100000000.0;
    for (j=i; j<= icnt; j++)
    {
      dist = sqrt(pow((nodes[n1][1] - nodes[vector[j]][1]), 2.0) +
pow((nodes[n1][2] - nodes[vector[j]][2]), 2.0));
    //  printf("n1 %d, n2 %d, j= %d, dist = %f; icnt = %d\n",n1, n2, j,
dist, icnt);
      if (dist < distmin)
      {
        iprov = vector[i];
        vector[i] = vector[j];
        vector[j] = iprov;
        distmin = dist;
      }
    }
  }
  for (j=2; j<= icnt; j++)
  {

   // fprintf(stdout, "%4d %4d %4d ", elmtcount, vector[j-1], vector[j]);
    //fprintf(stdout, "%10e %10e %10e %10e %5e\n",trans, aperture, dist,
stor, 0.0);
   // if (vector[j]!=vector[j-1])  might not need this
    {
      elements[elmtcount][1]=vector[j-1];    // start node
      elements[elmtcount][2]=vector[j];  // end node
      if (!node_conversion[vector[j-1]])  node_conversion[vector[j-
1]]=nodecount++;
      if (!node_conversion[vector[j]])    node_conversion[vector[j]]
=nodecount++;
      elmtcount++; // counts one too many, reset in main
    }
}
}

/*
```

```
void output_nodes()
/* This output loops writes the nodes to the output file in a format that
can be understood by the simulated annealing code */
{
  int i;
    float x,y,z;

  fprintf(nodefile,"          %5d\n",nodecount);
  fprintf(stdout,"nodes:    %5d\n",nodecount);

  for(i=1; i<= nodecount; i++)
  {
    x = nodes[node_conversion[i]][1];
    y = nodes[node_conversion[i]][2];
    z = 0.0;
    fprintf(nodefile,"%5d  0 0 0% 12lf% 12lf% 12lf% 12lf% 12lf\n", i,
x,y,z, head, 0.0);
    // fprintf(stdout,"nd %5d 0 0 0 %10f %10f %10f %10f %10f\n", ncnt,
x,y,z, head, 0.0);

  }
}
*/

void generate_border()
/* this generates the bonds around the edge of the figure. Needed because
the inverse code has to operate with a fixed head boundary. Note that the
sides of the quadrilateral have to be parallel to the x and y axis in order
for this to work!!! */
{
  int i,j,k, ifrac, node, lastnode;
  // lastnode is to remember last node in chain of linking border
  // it also contains the first (corner) node.
  double maxx, maxy, minx, miny, xx, yy,deltax, deltay;

  if (xcoord[1]==xcoord[2])
  {
    if(xcoord[1]<xcoord[3])
    {
      minx=xcoord[1];
      maxx=xcoord[3];
    }
    else
    {
      minx=xcoord[3];
      maxx=xcoord[1];
    }
  }
  else
  {
    if(xcoord[1]<xcoord[2])
    {
      minx=xcoord[1];
      maxx=xcoord[2];
    }
    else
    {
      minx=xcoord[2];
      maxx=xcoord[1];
    }
  }
  if (ycoord[1]==ycoord[2])
  {
```

```
    if(ycoord[1]<ycoord[3])
    {
      miny=ycoord[1];
      maxy=ycoord[3];
    }
    else
    {
      miny=ycoord[3];
      maxy=ycoord[1];
    }
  }
  else
  {
    if(ycoord[1]<ycoord[2])
    {
      miny=xcoord[1];
      maxy=xcoord[2];
    }
    else
    {
      miny=ycoord[2];
      maxy=ycoord[1];
    }
  }
  deltax=(maxx-minx)/(1<<(ilwctof));
  deltay=(maxy-miny)/(1<<(ilwctof));

  printf("minmax x, minmax y: %f,%f,%f,%f.\n", minx, maxx, miny, maxy);
  for (i=1; i<=2; i++)
  {
    node=i;
    xx=nodes[node][1];
    yy=nodes[node][2];
    if(xx>(maxx-deltax))   // compare coordinates of node with initial
corners
    {
      rightnodes[nright++]=node; // right edge node
      node_isc[node]=4;
      node_ib[node]=bound_type;
     // printf("adding rightnode %d:%f,%f\n", node, xx,yy);
    }
    if(xx<(minx+deltax))
    {
      leftnodes[nleft++]=node;  // left edge node
      node_isc[node]=2;
      node_ib[node]=bound_type;
     // printf("adding leftnode %d:%f,%f\n", node, xx,yy);
    }
    if(yy>(maxy-deltay))
    {
      topnodes[ntop++]=node; //top edge node
      node_isc[node]=3;
      node_ib[node]=bound_type;
     // printf("adding topnode %d:%f,%f\n", node, xx,yy);
    }
    if(yy<(miny+deltay))
    {
      bottomnodes[nbottom++]=node;     // bottom edge node
      node_isc[node]=1;
      node_ib[node]=bound_type;
     // printf("adding bottomnode %d:%f,%f\n", node, xx,yy);
    }
  }
```

```
  ifrac = 1;


  for (i=1; i<= ilwctof; i++)
  {
    for(j=1; j<=(1<<i); j++)
    {
      ifrac++;
      if (mafrac[ifrac] == 1)
      {
        for (k=1; k<=2; k++)
        {
          node=2*ifrac+k;
          xx=nodes[node][1];
          yy=nodes[node][2];
          if(xx>(maxx-deltax))   // compare coordinates of node with
initial corners
          {
            rightnodes[nright++]=node; // right edge node
            node_isc[node]=4;
            node_ib[node]=bound_type;
          //  printf("adding rightnode %d:%f,%f\n", node, xx,yy);

          }
          if(xx<(minx+deltax))
          {
            leftnodes[nleft++]=node;  // left edge node
            node_isc[node]=2;
            node_ib[node]=bound_type;
           // printf("adding leftnode %d:%f,%f\n", node, xx,yy);
          }
          if(yy>(maxy-deltay))
          {
            topnodes[ntop++]=node; //top edge node
            node_isc[node]=3;
            node_ib[node]=bound_type;
          //  printf("adding topnode %d:%f,%f\n", node, xx,yy);
          }
          if(yy<(miny+deltay))
          {
            bottomnodes[nbottom++]=node;     // bottom edge node
            node_isc[node]=1;
            node_ib[node]=bound_type;
          //  printf("adding bottomnode %d:%f,%f\n", node, xx,yy);
          }
        }
      }
    }
  }
  node_isc[(MAXNODES-4+0)]=1;  // bottom left (minx, miny)
  node_isc[(MAXNODES-4+1)]=1;
  node_isc[(MAXNODES-4+2)]=3;  // topright (maxx, maxy)
  node_isc[(MAXNODES-4+3)]=3;

  node_ib[(MAXNODES-4+0)]=bound_type;  // bottom left (minx, miny)
  node_ib[(MAXNODES-4+1)]=bound_type;
  node_ib[(MAXNODES-4+2)]=bound_type;  // topright (maxx, maxy)
  node_ib[(MAXNODES-4+3)]=bound_type;

  topnodes[ntop]=(MAXNODES-4+2);  // topright corner added
  rightnodes[nright]=(MAXNODES-4+2); // topright
  bottomnodes[nbottom]=(MAXNODES-4+1); // bottomright
```

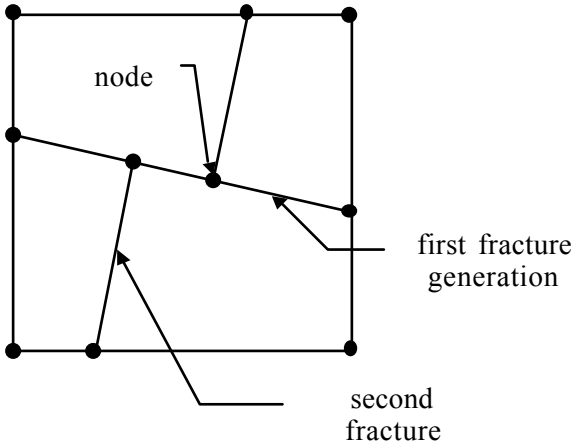
```
  leftnodes[nleft]=(MAXNODES-4+3); // top left
/* the following calls quick-sort the nodes and connections in order of
increasing x/y coordinates */
  qsort(&rightnodes[1],(nright), sizeof(rightnodes[0]), comparey); // dont
sort last corner node
  qsort(&leftnodes[1],(nleft), sizeof(rightnodes[0]), comparey);
  qsort(&bottomnodes[1],(nbottom), sizeof(rightnodes[0]), comparex);
  qsort(&topnodes[1],(ntop), sizeof(rightnodes[0]), comparex);

  lastnode=(MAXNODES-4+0); // i.e. bottom left node (minx, miny)
  for (i=1; i<=nbottom; i++)
  {
    elements[elmtcount][1]=lastnode;
    elements[elmtcount][2]=bottomnodes[i];
    lastnode=bottomnodes[i];
    elmtcount++;
  }
  lastnode=(MAXNODES-4+1); // i.e. bottom right node
  for (i=1; i<=nright; i++)
  {
    elements[elmtcount][1]=lastnode;
    elements[elmtcount][2]=rightnodes[i];
    lastnode=rightnodes[i];
    elmtcount++;
  }
  lastnode=(MAXNODES-4+3); // i.e. top left
  for (i=1; i<=ntop; i++)
  {
    elements[elmtcount][1]=lastnode;
    elements[elmtcount][2]=topnodes[i];
    lastnode=topnodes[i];
    elmtcount++;
  }
  lastnode=(MAXNODES-4+0); // i.e. bottom left node
  for (i=1; i<=nleft; i++)
  {
    elements[elmtcount][1]=lastnode;
    elements[elmtcount][2]=leftnodes[i];
    lastnode=leftnodes[i];
    elmtcount++;
  }
  elmtcount--;
}


static int comparey(const int *nodea, const int *nodeb)
{
  if (nodes[*nodea][2]>nodes[*nodeb][2])
    return(1);
  if (nodes[*nodea][2]<nodes[*nodeb][2])
    return(-1);
  return(0);
}
static int comparex(const int *nodea, const int *nodeb)
{
  if (nodes[*nodea][1]>nodes[*nodeb][1])
    return(1);
  if (nodes[*nodea][1]<nodes[*nodeb][1])
    return(-1);
  return(0);
}
```

```
void output()
{
  int i;
  int nodecounter=1;
  int node1, node2;
  double dist;
  double x,y,z;

  fprintf(elmtfile, "          %5d\n",elmtcount);
// writes the element file (i.e. starting and endnode and add. info that
// is needed for the inverse code
  fprintf(xyzfile,  "               %5d\n",elmtcount);
// writes the xyz co-ordinates of all start and end-nodes
// this can be used to plot the network
  fprintf(stdout,"elements:    %5d\n",elmtcount);

  fprintf(nodefile,"          %5d\n",nodecount);
  fprintf(nodexyzfile,"          %5d\n",nodecount);
  fprintf(stdout,"nodes:    %5d\n",nodecount);

  for(i=1; i<= elmtcount; i++)
  {
    dist  = pow((nodes[elements[i][1]][1] - nodes[elements[i][2]][1]),
2.0);
    dist += pow((nodes[elements[i][1]][2] - nodes[elements[i][2]][2]),
2.0);
    dist = sqrt(dist);
    fprintf(elmtfile, "%5d%5d%5d", i, node_conversion[elements[i][1]],
node_conversion[elements[i][2]]);
    fprintf(elmtfile, " %.5e %.5e %.5e %.5e %.5e\n",trans, aperture, dist,
stor, 0.0);
    node1=elements[i][1];
    node2=elements[i][2];
    if (node_conversion[elements[i][1]]>= nodecounter)
    {
      x = nodes[node1][1];
      y = nodes[node1][2];
      z = 0.0;
      fprintf(nodefile,"%5d%3d%2d 0 % .4e % .4e % .4e % .4e % .4e\n",
nodecounter++,node_isc[node1],node_ib[node1], x,y,z, head, 0.0);
      fprintf(nodexyzfile,"%10f %10f\n",  x,y); // maybe extend to 3D
    }
    if (node_conversion[elements[i][2]]>= nodecounter)
    {
      x = nodes[node2][1];
      y = nodes[node2][2];
      z = 0.0;
      fprintf(nodefile,"%5d%3d%2d 0 % .4e % .4e % .4e % .4e % .4e\n",
nodecounter++,node_isc[node2],node_ib[node2], x,y,z, head, 0.0);
      fprintf(nodexyzfile,"%10f %10f\n",  x,y); // maybe extend to 3D

    }
    fprintf(xyzfile,"% 12e % 12e % 12e % 12e\n", nodes[node1][1],
nodes[node1][2], nodes[node2][1],nodes[node2][2]);

  }
}
```

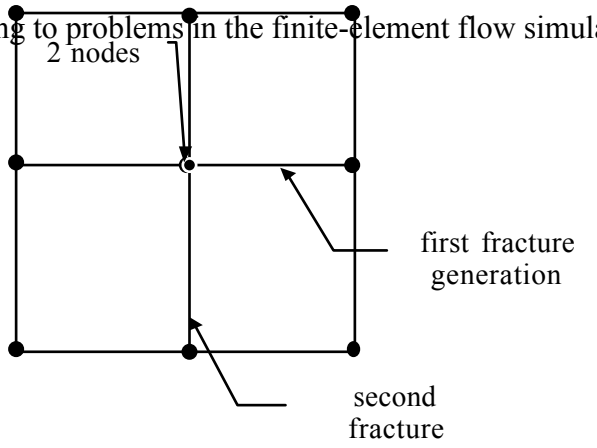
Appendix C - Post-processing of fracture network files

Two problems were encountered by the author that require post-processing of the generated fracture networks in order to use them in the used finite-element flow simulation and inverse modelling code.

1. The iterated function system approach results in the generation of two new fractures with their associated start and end nodes from each fracture one generation above as shown below:



A problem arises, when a network geometry is chosen such that the initial fracture is parallel to either side of the original network. Then, quadratic fracture blocks are generated and for example, fractures of the second generation will touch at one point on the fracture of the previous generation. This leads to nodes with the same co-ordinates, and a bond length close to zero, leading to problems in the finite-element flow simulation. This is illustrated below.



A programme was written to eliminate such nodes, which in turn requires renumbering of all nodes and bonds. The programme itself is reasonably trivial and is only included on disk. If the initial fracture is very oblique, this can also lead to multiple nodes that are very close to each other and result in very short bond-lengths. Hence it is suggested that this post-processing is performed for all generated networks.

2. A second, more severe problem is described here. The representation of the network required by the finite-element code consists of a node file and an element file. The node file describes the co-ordinates of each node. Each node has a number and the element file then defines each bond by the numbers of the start- and the end-node. The memory requirements and the efficiency of the finite-element code depend critically on the bandwidth of the network, which is defined by the largest difference in node number between any nodes that are connected by a bond. For large networks, the number of nodes can be of the order of several thousands, which means that it is very likely that two nodes that are connected have a difference in node number of the same order. To reduce the memory requirements for such networks and to allow execution of the code in the first place, it is necessary to perform a bandwidth reduction, which tries to renumber all nodes such that a near-minimum bandwidth is achieved.
   A programme that does this is shown below. The algorithm used was presented in the two references given below, and rights of the algorithm also belong to the authors.

- Gibbs, N.E., Poole, W.G., Stockmeyer, P.K., 1976, An algorithm for reducing the bandwidth and profile of a sparse matrix, *SIAM Journal of Numerical Analysis*, **13**, 2, 236-250
- Crane, H.L., Gibbs, N.E., Poole, W.G., Stockmeyer, P.K., 1976, Algorithm 508:Matrix Bandwidth and Profile Reduction [FI], ACM Transactions on mathematical Software, **2**, 4, 374-377

```
//#include <console.h>        // include these lines when compiling
//#include <SIOUX.h>          // on Macintosh Codewarrior

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>

#define MAXCONNEC 30          // maximum number of disconnected domains
#define MAXNODES (1<<16+1)    // maximum number of nodes
#define MAXLAST 2000+1
#define MAXCOMPS 500+1
// Global Variables


static int ndstk[MAXNODES][MAXCONNEC]; // stores connection table (input)
//int nr=MAXCONNEC;           // max number of nodes        (input)
int n;                        // number of nodes                 (input)
static int iold[MAXNODES];    // old numbering of nodes     (input)
static int renum[(MAXNODES+1)];// renumbered nodes               (output)
static int ndeg[MAXNODES];          // degree of each node       (output)
static int lvl[MAXNODES];
static int lvls1[MAXNODES];         // node numbers listed ny level
static int lvls2[MAXNODES];         // level assigned to node i by reduce
static int ccstor[MAXNODES];  // storage array
int idpth;                    // number of levels in reduce level structure
int ideg;                     // max degree of any node

int lvln, lvlbot, lvlwth, lroot, maxlw;

static int nhigh[MAXLAST], nlow[MAXLAST], nacum[MAXLAST];
static int ndlst[MAXLAST];
static int stkd[MAXLAST];
                              // assumes that there are at most
                              // MAXLAST nodes in the last level
int xc, size[MAXCOMPS], stpt[MAXCOMPS];
            // assumes there are at most MAXCOMPS connected components
int ibw1, ibw2, ipf1, ipf2;         // output vars for bandwidth and
profile after renumb.
int isdir, num;
// on return from piklvl, isdir indicates the direction the largest
// component cell.  Isdir is modified now to indicate the numbering
// direction.  Num is set to the proper value for this direction.
int oldbw;                    // stores old bandwidth (computed)
int nflg;
int idflt;
int node_con[MAXNODES+1];     // node number conversion: reorders according
to renum


/* end of reduce globals, following the file operation/ element&node admin
vars*/


static struct elem {    /* array of element information */
  int num;       /* not used */
  int index;     /* (inp./det. int.) = 0 -> element off, = 1 -> on */
  int n[2];      /* (inp.) numbers of nodes connected to element */
  double transm; /* (inp.) transmissivity (units?) */
  double apert;  /* (inp.) aperture (units?) */
  double length; /* (inp.) length (units?) */
  double stor;   /* (inp.) storativity (units?) */
  double diff;   /* (inp.) diffusivity (units?) */
```

```
  double dist;   /* (calc.) distance from the origin (units?) */
  int next[2];   /* (det. int.) next[i] is the number of another
                    element connected to node n[i] */
} *elems;
int nelmts;

static struct node {    /* array of node information */
  double x,y,z;  /* (inp.) coordinates (units?) */
  double bvalu;  /* (inp.) boundary value (units?) */
  double hbv;    /* (inp.) head boundary value (units?) */
  double fbv;    /* (inp.) flow boundary value (units?) */
  double cbvalu; /* *** read, but not used *** */
  int isc;       /* (inp.) number of side on which node lies; = 0 if node
is an internal node */
  int ib;        /* (inp./ det. int.) boundary type (= ob below) */
  int ob;        /* (inp.) original boundary type */
  int elem;      /* (det. int.) number of an element connected to this node
*/
  int hbi;       /* (det. int.) index of this node in array hb */
  int fbi;       /* (det. int.) index of this node in array fb */
} *nodes;
int nnodes;
char elmt_in[100];
char node_in[100];
char elmt_out[100];
char node_out[100];



/* Prototypes for reduce */
void reduce(void);
void dgree(void);
void fndiam(int *, int *, int *);
void tree(int ,int *, int);
void sortdg(int *, int *, int *, int *);
void setup(void);
int sort2(void);
void piklvl(void);
void number(int *, int *, int *, int *);
/* end of prototypes */

/* prototypes for reading data */
void read_data(void);
void init_data(void);
void max_con(void);
void write_data();
void relmt(FILE *);
void rnode(FILE *);
void welmt(FILE *);
void wnode(FILE *);

/* end of prototypes */


void main(int argc, char **argv)
{
  //argc = ccommand(&argv);         // Use this line if using with
                                    // Metroworks Codewarrior on Macintosh
  if (argc==1)
  {
    fprintf(stdout,"  This program tries to reduce the bandwidth for a
given node and element file.\n");
```

```
    fprintf(stdout,"  Each element is a line, defined by the endpoint
coordinates given by two nodes.\n");
    fprintf(stdout,"  The numbering scheme of these two nodes defines the
bandwidth of a sparse matrix\n");
    fprintf(stdout,"  whose items are non zero for each element {i,j} where
i and j are node numbers.\n");


    fprintf(stdout,"  USAGE: reduce elmt_in node_in elmt_out node_out\n");
    fprintf(stdout,"  If first argument is 'default', fracelmt.inp,
fracnode.inp, elmt.sol and node.inp are used.\n");
    fprintf(stdout,"  for queries contact Heiko Paelike.\n");
    exit(1);
  }
  else if ((argc==2) && (strcmp(argv[1],"default")==0))
  {
    fprintf(stdout, "  Using default file names elmt.inp, node.inp and
elmtxyz.txt .\n");
    strcpy(elmt_in,"fracelmt.inp");
    strcpy(node_in,"fracnode.inp");
    strcpy(elmt_out,"elmt.sol");
    strcpy(node_out,"node.inp");

  }
  else if(argc==5)
  {
    strcpy(elmt_in,argv[1]);
    strcpy(node_in,argv[2]);
    strcpy(elmt_out,argv[3]);
    strcpy(node_out,argv[4]);
  }
  else if (argc!=5)
  {
    fprintf(stdout, "Wrong number of arguments, try 'reduce' (no arguments)
for usage.\n");
    exit(1);
  }
  fprintf(stdout, "entering read_data()\n");
  read_data();
  fprintf(stdout, "entering init_data()\n");
  init_data();
  fprintf(stdout, "entering reduce()\n");
  reduce();
  fprintf(stdout, "entering write_data()\n");
  write_data();
  fprintf(stdout, "Execution complete.\n");
  exit(0);
}

void reduce(void)
{
  int  sbnum;           // sbnum= low end of available numbers for
renumbering
  int  stnum;           // stnum= high end of available numbers for
renumbering

  int lowdg;
  static int stnode, rvnode;

  int i;                // counting variable

//  subroutine reduce determines a row and column permutation which,
//  when applied to a given sparse matrix, produces a permuted
```

```
//  matrix with a smaller bandwidth and profile.
//  The input array is a connection table which represents the
//  indices of the nonzero elements of the matrix, a.  The algo-
//  rithm is described in terms of the adjacency graph which
//  has the characteristics
//  that there is an edge (connection)
//  between nodes I and j if a(i,j) .Ne. 0 and I .Ne. J.
//  Dimensioning information--the following integer arrays must be
//  dimensioned in the calling routine.
//    Ndstk(nr,d1)        d1 is .Ge. Maximum degree of all nodes.
//    Iold(d2)            d2 and nr are .Ge. The total number of
//    renum(d2+1)         nodes in the graph.
//    ndeg(d2)            storage requirements can be significantly
//    lvl(d2)             decreased for ibm 360 and 370 computers
//    lvls1(d2)           by replacing integer ndstk by
//    lvls2(d2)           integer*2 ndstk in subroutines reduce,
//    ccstor(d2)          dgree, fndiam, tree and number.
//  Common information--the following common block must be in the
//  calling routine.
//    Common/gra/n,idpth,ideg
//  explanation of input variables--
//    ndstk-     connection table representing graph.
//               Ndstk(i,j)=node number of jth connection to node
//               number i.  A connection of a node to itself is not
//               listed.  Extra positions must have zero fill.
//    Nr-        row dimension assigned ndstk in calling program.
//    Iold(i)-   numbering of ith node upon input.
//               If no numbering exists then iold(i)=i.
//    N-         number of nodes in graph (equal to order of matrix).
//    Ideg-      maximum degree of any node in the graph.
//  Explanation of output variables--
//    renum(i)-  the new number for the ith node.
//    Ndeg(i)-   the degree of the ith node.
//    Ibw2-      the bandwidth after renumbering.
//    Ipf2-      the profile after renumbering.
//    Idpth-     number of levels in reduce level structure.
//  The following only have meaning if the graph was connected--
//    lvl(i)-    index into lvls1 to the first node in level i.
//               Lvl(i+1)-lvl(i)= number of nodes in ith level
//    lvls1-     node numbers listed by level.
//    Lvls2(i)-  the level assigned to node I by reduce.
//  Working storage variable--
//    ccstor
//  local storage--
//    common/cc/-subroutines reduce, sort2 and piklvl assume that
//               the graph has at most MAXCOMPS connected components.
//               Subroutine fndiam assumes that there are at most
//               MAXLAST nodes in the last level.
//    Common/lvlw/-subroutines setup and piklvl assume that there
//               are at most MAXLAST levels.
// Use integer*2 ndstk  with an ibm 360 or 370.
// it is assumed that the graph has at most MAXCOMPS connected components.
  ibw2 = 0;
  ipf2 = 0;
// set renum(i)=0 for all I to indicate node I is unnumbered
  for (i=1; i<=n; i++)
    renum[i] = 0;
// compute degree of each node and original bandwidth and profile
//  dgree(ndstk, nr, ndeg, iold, ibw1, ipf1)
  fprintf(stdout, "entereing dgree to compute connections of each node\n");
  dgree();
// sbnum= low end of available numbers for renumbering
// stnum= high end of available numbers for renumbering
```

```
  sbnum = 1;
  stnum = n;
  fprintf(stdout, "number nodes of degree zero.\n");
// number the nodes of degree zero
  for (i=1; i<=n; i++)
  {
    if (ndeg[i]>0) continue;
    renum[i] = stnum;
    stnum--;
  }
fprintf(stdout, "find unnumbered node of min degree to start on ...\n");
// find an unnumbered node of min degree to start on
// 30
  do
  {
     lowdg = ideg + 1;
       nflg = 1;
       isdir = 1;
       for (i=1; i<=n; i++)
       {
         if (ndeg[i]>=lowdg) continue;
         if (renum[i]>0) continue;
         lowdg = ndeg[i];
         stnode = i;
     }
fprintf(stdout, "this node is numbered %d.\n", stnode);
// find pseudo-diameter and associated level structures.
// Stnode and rvnode are the ends of the diam and lvls1 and lvls2
// are the respective level structures.
//     fndiam(&stnode, &rvnode, &ndstk, nr, &ndeg, &lvl, &lvls1, &lvls2,
&ccstor, ??? idflt);

fprintf(stdout,"finding pseudo-diameter, entering fndiam\n");
     fndiam(&stnode, &rvnode,ccstor);
     if (ndeg[stnode]>ndeg[rvnode])
       {
// nflg indicates the end to begin numbering on
          nflg = -1;
          stnode = rvnode;
       }
//50
    // setup(lvl, lvls1, lvls2);
fprintf(stdout,"setup ...\n");
     setup();
// find all the connected components  (xc counts them)
     xc = 0;
       lroot = 1;
       lvln = 1;
       for (i=1; i<=n; i++)
     {
         if (lvl[i]==0)
         {
           xc++;
           stpt[xc] = lroot;
           tree(i,ccstor,n);
           size[xc] = lvlbot + lvlwth - lroot;
           lroot = lvlbot + lvlwth;
           lvln = lroot;
         }
       }
   fprintf(stdout, "xc =%d. now calling piklvl()\n", xc);
       if (sort2()!=0)
       {
```

```
         piklvl();
           fprintf(stdout,"exited piklvl\n");
         }
// on return from piklvl, isdir indicates the direction the largest
// component fell.  Isdir is modified now to indicate the numbering
// direction.  Num is set to the proper value for this direction.
       isdir = isdir*nflg;
       num = sbnum;
     if (isdir<0) num = stnum;
fprintf(stdout, "entering number\n");
       number(&stnode, lvls1, lvl, ccstor);
// update stnum or sbnum after numbering
       if (isdir<0) stnum = num;
       if (isdir>0) sbnum = num;
  }
  while (sbnum<=stnum);
  if (ibw2<=ibw1)
  {
    printf ("old bandwidth = %d. new bandwidth = %d.Exit\n", oldbw, ibw2);
    return;
  }
// if original numbering is better than new one, set up to return it
  printf("old numbering better was better. no change.\n");
  for (i=1; i<=n;i++)
    renum[i] = iold[i];
  ibw2 = ibw1;
  ipf2 = ipf1;
  return;
}



void dgree(void)
{
    //  subroutine dgree(ndstk, nr, ndeg, iold, ibw1, ipf1)
dgr   10
//  dgree computes the degree of each node in ndstk and stores
//  it in the array ndeg.  The bandwidth and profile for the original
//  or input renumbering of the graph is computed also.
// Use integer*2 ndstk  with an ibm 360 or 370.
    //  Integer ndstk
    //  common /gra/ n, idpth, ideg
    //  dimension ndstk(nr,1), ndeg(1), iold(1)
  int i,j;
  int itst;
  int irw, idif;
  int maxdg=0;

  ibw1 = ipf1 = 0;
  for (i=1; i<= n; i++)
  {
    ndeg[i]=0;
    irw = 0;
    for (j=1; j<= ideg; j++)
    {
      itst = ndstk[i][j];
      if (itst<=0) break;
    // 10
      ndeg[i]++;
      if (ndeg[i]>maxdg) maxdg=ndeg[i];
      idif = iold[i] - iold[itst];
      if (irw < idif) irw=idif;
    }
```

```
    //30
    ipf1 += irw;
    if (irw>ibw1) ibw1=irw;
  }
}



void fndiam(int *snd1, int *snd2, int *iwk)
{
  int ndxl, ndxn;
  int flag, i;
  int snd;
  int mtw1, mtw2;
  int *ptr;

   //   subroutine fndiam(snd1, snd2, ndstk, nr, ndeg, lvl, lvls1,
fnd   10
   //  * lvls2, iwk, idflt)
//  fndiam is the control procedure for finding the pseudo-diameter of
//  ndstk as well as the level structure from each end
//  snd1-        on input this is the node number of the first
//               attempt at finding a diameter.  On output it
//               contains the actual number used.
//  snd2-        on output contains other end of diameter
//  lvls1-       array containing level structure with snd1 as root
//  lvls2-       array containing level structure with snd2 as root
//  idflt-       flag used in picking final level structure, set
//               =1 if width of lvls1 .Le. Width of lvls2, otherwise =2
//  lvl,iwk-     working storage
// use integer*2 ndstk  with an ibm 360 or 370.
    //  Integer ndstk
    //  integer flag, snd, snd1, snd2
    //  common /gra/ n, idpth, ideg
// it is assumed that the last level has at most MAXLAST nodes.
    //  Common /cc/ ndlst(MAXLAST)
    //  dimension ndstk(nr,1), ndeg(1), lvl(1), lvls1(1), lvls2(1),
    // * iwk(1)
  mtw1 = mtw2 = 0;
  flag = 0;
  mtw2 = n;
  snd = *snd1;
  fprintf(stdout, "snd = %d. n= %d\n", snd,n);
// zero lvl to indicate all nodes are available to tree
//10
  do
  {
        do
        {
          for (i=1; i<=n; i++)
            lvl[i] = 0;
          lvln = 1;
      // drop a tree from snd
            fprintf(stdout, "dropping tree with tree(), snd = %d\n",snd);
          tree(snd,iwk, mtw2);
          if (flag>=1)
          {
            if (idpth>=(lvln-1)) break;
      // start again with new starting node
            *snd1 = snd;
          }
          else flag = 1;
      //30
```

```
          idpth = lvln - 1;
          mtw1 = maxlw;
      // copy level structure into lvls1
          for(i=1;i<=n;i++)
            lvls1[i] = lvl[i];
          ndxn = 1;
          ndxl = 0;
          mtw2 = n;
      // sort last level by degree  and store in ndlst
          ptr = &(iwk[lvlbot]);
          ptr--; // index adjustment
            fprintf(stdout,"calling sortdg with ndlst[0] = %d\n",ndlst[0]);
          sortdg(&ndlst[0], ptr, &ndxl, &lvlwth);
          snd = ndlst[1];
        }
        while (1);
      //60
        if (maxlw<mtw2)
        {
          mtw2 = maxlw;
          *snd2 = snd;
      // store narrowest reverse level structure in lvls2
          for(i=1;i<=n;i++)
            lvls2[i] = lvl[i];
        }
    if (ndxn==ndxl) break;
// try next node in ndlst
    ndxn++;
    snd = ndlst[ndxn];
  }
  while(1);
  idflt = 1;
  if (mtw2<=mtw1) idflt = 2;
}



void tree(int iroot,int *iwk, int ibort)  // checked, OK
{
  int j;
  int iwknow, inow, itop, itest, lvltop, ndrow;
//  tree drops a tree in ndstk from iroot
//  lvl-         array indicating available nodes in ndstk with zero
//               entries. Tree enters level numbers assigned
//               during execution of this procedure
//  iwk-         on output contains node numbers used in tree
//               arranged by levels (iwk(lvln) contains iroot
//               and iwk(lvlbot+Lvlwth-1) contains last node entered)
//  lvlwth-      on output contains width of last level
//  lvlbot-      on output contains index into iwk of first
//               node in last level
//  maxlw-       on output contains the maximum level width
//  lvln-        on input the first available location in iwk
//               usually one but if iwk is used to store previous
//               connected components, lvln is next available location.
//               On output the total number of levels + 1
//  ibort-       input param which triggers early return if
//               maxlw becomes .Ge. Ibort
// use integer*2 ndstk  with an ibm 360 or 370.
   // Integer ndstk
   // dimension ndstk(nr,1), lvl(1), iwk(1), ndeg(1)
  maxlw = 0;
```

```
  itop = lvln;
  inow = lvln;
  lvlbot = lvln;
  lvltop = lvln + 1;
  lvln = 1;
  lvl[iroot] = 1;
  iwk[itop] = iroot;
// 10
  do
  {
        lvln++;
      // 20
        do
        {
              iwknow = iwk[inow];
              ndrow = ndeg[iwknow];
              for (j=1; j<=ndrow; j++)
              {
                itest = ndstk[iwknow][j];
                if (lvl[itest] != 0) continue;
                lvl[itest] = lvln;
                itop++;
                iwk[itop] = itest;
             //30
              }
              inow++;
            }
            while (inow<lvltop);
            lvlwth = lvltop - lvlbot;
        if (maxlw < lvlwth) maxlw = lvlwth;
        if (maxlw >= ibort) return;
        if (itop<lvltop) return;
        lvlbot = inow;
        lvltop = itop + 1;
  }
  while (1);
}
```

```
void sortdg(int *stk1, int *stk2, int *x1, int *x2)  // checked: OK!
{
  int i, j;
  int ind;
  int istk2, jstk2;
  int temp;
  int itest;
// sortdg sorts stk2 by degree of the node and adds it to the end
// of stk1 in order of lowest to highest degree.  x1 and x2 are the
// number of nodes in stk1 and stk2 respectively.
  //      Integer x1, x2, stk1, stk2, temp
  //    common /gra/ n, idpth, ideg
  //    dimension ndeg(1), stk1(1), stk2(1)
      ind = *x2;
// 10
  do
  {
            itest = 0;
            ind--;
            if (ind<1)
            {
              for (i=1; i<=(*x2); i++)
```

```
              {
                (*x1)++;
                stk1[*x1] = stk2[i];
              }
              return;
            }
              for (i=1; i<=ind; i++)
              {
                j = i + 1;
                istk2 = stk2[i];
                jstk2 = stk2[j];
                if (ndeg[istk2] <= ndeg[jstk2]) continue;
                itest = 1;
                temp = stk2[i];
                stk2[i] = stk2[j];
                stk2[j] = temp;
              }
        }
        while (itest==1);
        for (i=1; i<=*x2; i++)
        {
      (*x1)++;
      stk1[*x1] = stk2[i];
    }
    return;
}
```

```
void setup(void)
{
  int itemp, i;
// setup computes the reverse leveling info from lvls2 and stores
// it into lvls2.  Nacum(i) is initialized to nodes/ith level for nodes
// on the pseudo-diameter of the graph.  Lvl is initialized to non-
// zero for nodes on the pseudo-diam and nodes in a different
// component of the graph.
     // Common /gra/ n, idpth, ideg
// it is assumed that there are at most MAXLAST levels.
     // Common /lvlw/ nhigh(MAXLAST), nlow(MAXLAST), nacum(MAXLAST)
     // dimension lvl(1), lvls1(1), lvls2(1)
  for(i=1; i<= idpth; i++)
    nacum[i] = 0;
  for(i=1; i<= n; i++)
  {
    lvl[i] = 1;
    lvls2[i] = idpth + 1 - lvls2[i];
    itemp = lvls2[i];
    if (itemp>idpth) continue;
    if (itemp!=lvls1[i])
    {
      lvl[i]=0;
      continue;
    }
    nacum[itemp]++;
  }
}

int sort2(void)
{
  int i,j, ind;
```

```
  int temp, itest;
  int sort2ret;

// sort2 sorts size and stpt into descending order according to
// values of size. Xc=number of entries in each array
  // INTEGER temp, ccstor, size, stpt, xc;
// it is assumed that the graph has at most MAXCOMPS connected components.
  // Common /cc/ xc, size(MAXCOMPS), stpt(MAXCOMPS)
  sort2ret = 0;
  if (xc==0) return(sort2ret);
  sort2ret = 1;
  ind = xc;
//10
  do
  {
    itest = 0;
    ind--;
    if (ind<1) return(sort2ret);
    for (i=1;i<=ind;i++)
    {
      j = i + 1;
      if (size[i]>=size[j]) continue;
      itest = 1;
      temp = size[i];
      size[i] = size[j];
      size[j] = temp;
      temp = stpt[i];
      stpt[i] = stpt[j];
      stpt[j] = temp;
    }
  }
  while (itest==1);
  return(sort2ret);
}

void piklvl(void)
{
// piklvl chooses the level structure  used in numbering graph
// lvls1-       on input contains forward leveling info
// lvls2-       on input contains reverse leveling info
//              on output the final level structure chosen
// ccstor-      on input contains connected component info
// idflt-       on input =1 if wdth lvls1.Le.Wdth lvls2, =2 otherwise
// nhigh        keeps track of level widths for high numbering
// nlow-        keeps track of level widths for low numbering
// nacum-       keeps track of level widths for chosen level structure
// xc-          number of connected components
// size(i)-     size of ith connected component
// stpt(i)-     index into ccstore of 1st node in ith con compt
// isdir-       flag which indicates which way the largest connected
//              component fell.  =+1 if low and -1 if high
  //    integer ccstor, size, stpt, xc, end
  //    common /gra/ n, idpth, ideg
// it is assumed that the graph has at most MAXCOMPS components and
// that there are at most MAXLAST levels.
  //    Common /lvlw/ nhigh(MAXLAST), nlow(MAXLAST), nacum(MAXLAST)
  //    common /cc/ xc, size(MAXCOMPS), stpt(MAXCOMPS)
  //    dimension lvls1(1), lvls2(1), ccstor(1)

// for each connected component do
  int i,j,k, it;
  int end, inode, lvlnh, lvlnl;
  int max1, max2;
```

```
  for(i=1;i<=xc;i++)
  {
    j = stpt[i];
    end = size[i] + j - 1;
// set nhigh and nlow equal to nacum
    for (k=1; k<=idpth; k++)
    {
      nhigh[k] = nacum[k];
      nlow[k] = nacum[k];
    }
// update nhigh and nlow for each node in connected component
    for(k=j; k<=end; k++)
    {
      inode = ccstor[k];
      lvlnh = lvls1[inode];
      nhigh[lvlnh]++;
      lvlnl = lvls2[inode];
      nlow[lvlnl]++;
    }
    max1 = 0;
    max2 = 0;
// set max1=largest new number in nhigh
// set max2=largest new number in nlow
    for(k=1;k<=idpth;k++)
    {
      if ((2*nacum[k])==(nlow[k]+nhigh[k])) continue;
      if (nhigh[k]>max1) max1 = nhigh[k];
      if (nlow[k]>max2) max2 = nlow[k];
    }
// set it= number of level structure to be used
    it = 1;
    if (max1>max2) it = 2;
    if (max1==max2) it = idflt;
    if (it==2)
    {
      for(k=1;k<=idpth;k++)
        nacum[k] = nlow[k];
      continue;
    }
    if (i==1) isdir = -1;
// copy lvls1 into lvls2 for each node in connected component
    for (k=j; k<=end; k++)
    {
      inode = ccstor[k];
      lvls2[inode] = lvls1[inode];
    }
// update nacum to be the same as nhigh
    for (k=1; k<=idpth;k++)
      nacum[k] = nhigh[k];
// update nacum to be the same as nlow
//80
  }
  return;
}


void number(int *snd, int *lvlst, int *lstpt, int *ipfa)
{

 //     subroutine number(snd, num, ndstk, lvls2, ndeg, renum, lvlst,
num   10
 //    * lstpt, nr, nflg, ibw2, ipf2, ipfa, isdir)
```

```
//  number produces the numbering of the graph for min bandwidth
//  snd-         on input the node to begin numbering on
//  num-         on input and output, the next available number
//  lvls2-       the level structure to be used in numbering
//  renum-       the array used to store the new numbering
//  lvlst-       on output contains level structure
//  lstpt(i)-    on output, index into lvlst to first node in ith lvl
//               lstpt(i+1) - lstpt(i) = number of nodes in ith lvl
//  nflg-        =+1 if snd is forward end of pseudo-diam
//               =-1 if snd is reverse end of pseudo-diam
//  ibw2-        bandwidth of new numbering computed by number
//  ipf2-        profile of new numbering computed by number
//  ipfa-        working storage used to compute profile and bandwidth
//  isdir-       indicates step direction used in numbering(+1 or -1)
// use integer*2 ndstk  with an ibm 360 or 370.

      // Integer ndstk
  int i, j;
  int *stka, *stkb, *stkc;
  int xa, xb, xc, xd, cx, lst, lnd, ipro, inx;
  int end, test, max;
  int nstpt, nbw;

  stka=nhigh;
  stkb=nlow;
  stkc=nacum;
    // stkd made global
//  common /gra/ n, idpth, ideg
// the storage in common blocks cc and lvlw is now free and can
// be used for stacks.
    //  Common /lvlw/ stka(MAXLAST), stkb(MAXLAST), stkc(MAXLAST)
    //  common /cc/ stkd(MAXLAST)
    //  dimension ipfa(1)
    //  dimension ndstk(nr,1), lvls2(1), ndeg(1), renum(1), lvlst(1),
    // * lstpt(1)
// set up lvlst and lstpt from lvls2
  for (i=1; i<=n; i++)
    ipfa[i] = 0;
  nstpt = 1;
  for (i=1; i<=idpth; i++)
  {
    lstpt[i]=nstpt;
    for (j=1; j<=n; j++)
    {
      if (lvls2[j] == i)
      {
        lvlst[nstpt] = j;
        nstpt++;
      }
    }
  }
  lstpt[idpth+1] = nstpt;

// stka, stkb, stk// and stkd are stacks with pointers
// xa,xb,xc, and xd.  Cx is a special pointer into stkc which
// indicates the particular node being processed.
// Lvln keeps track of the level we are working at.
// Initially stkc contains only the initial node, snd.
      lvln = 0;
      if (nflg < 0) lvln = idpth + 1;
      xc=1;
      stkc[xc] = *snd;
```

```
// 40
  do
  {
    cx = 1;
    xd = 0;
    lvln += nflg;
    lst = lstpt[lvln];
    lnd = lstpt[lvln+1] - 1;
// begin processing node stkc(cx)
// 50
    do
    {
      ipro = stkc[cx];
      renum[ipro] = num;
      num += isdir;
      end = ndeg[ipro];
      xa = 0;
      xb = 0;
// check all adjacent nodes
      for (i=1; i<= end; i++)
      {
        test = ndstk[ipro][i];
        inx = renum[test];
// only nodes not numbered or already on a stack are added
        if (inx == 0)
        {
// put nodes on same level on stka, all others on stkb
          renum[test]= -1;
          if (lvls2[test] != lvls2[ipro])
          {
            xb++;
            stkb[xb] = test;
          }
          else
          {
            xa++;
            stka[xa]=test;
          }
        }
        else if (inx > 0)
        {
// do preliminary bandwidth and profile calculations
          nbw = (renum[ipro]-inx)*isdir;
          if (isdir > 0) inx = renum[ipro];
          if (ipfa[inx] < nbw) ipfa[inx] = nbw;
        }
      }
// 80
// sort stka and stkb into increasing degree and add stka to stkc
// and stkb to stkd


      if ((xa != 0) && (xa !=1))
      {
        sortdg(stkc, stka, &xc, &xa);
      }
      else if (xa == 1)
      {
        xc++;
        stkc[xc] = stka[xa];
      }

      if ((xb != 0) && (xb !=1))
```

```
      {
        sortdg(stkd, stkb, &xd, &xb);
      }
      else if (xb == 1)
      {
        xd++;
        stkd[xd] = stkb[xb];
      }
      cx++;
      if  (xc >= cx) continue;
// when stk// is exhausted look for min degree node in same level
// which has not been processed
      max = ideg + 1;
      *snd = n + 1;
      for (i=lst; i<= lnd; i++)
      {
        test = lvlst[i];
        if (renum[test] != 0) continue;
        if (ndeg[test] >=max) continue;
        renum[*snd] = 0;
        renum[test] = -1;
        max = ndeg[test];
        *snd = test;
// 130
      }
      if (*snd == (n+1))  break;
      xc++;
      stkc[xc] = *snd;
    }
    while (1);
// if stkd is empty we are done, otherwise copy stkd onto stkc
// and begin processing new stkc
    if (xd==0) break;
    for (i=1; i<= xd; i++)
      stkc[i] = stkd[i];
// 150
    xc = xd;
  }
  while (1);
// do final bandwidth and profile calculations
//  160
  for (i=1; i<=n; i++)
  {
    if (ipfa[i]>ibw2) ibw2=ipfa[i];
    ipf2 += ipfa[i];
  }
//  170
}


void init_data()
{
  int i;
  for (i=1; i<= nnodes; i++)
    iold[i] = i;
  max_con();  // determine maximum degree and set up connection matrix in
ndstk


}

void max_con(void)
{
```

```
  struct elem *ep;
  int i;
  int test, test1;
  int ne;
  int oldbwtest;
  int *nodeticks; // increased by one if element uses it.
  //printf("entered maxconn\n");


  if((nodeticks = (int *)calloc((nnodes+1),(unsigned)sizeof(int))) == NULL)
  {
    perror("max_con-calloc"); exit(1);
  }

  ep = elems;
  ideg = 0;
  oldbw= oldbwtest=0;

  for (ne = 0; ne < nelmts; ne++)
  {
    oldbwtest=fabs(ep->n[0]-ep->n[1]);
    if (oldbwtest>oldbw) oldbw=oldbwtest;
    test=nodeticks[ep->n[0]]++;
    test1=nodeticks[ep->n[1]]++;
    if ((test<MAXCONNEC) && (test1<MAXCONNEC))
    {
      ndstk[ep->n[0]][nodeticks[ep->n[0]]]=ep->n[1];
      ndstk[ep->n[1]][nodeticks[ep->n[1]]]=ep->n[0];
    }
    else
    {
      perror(" max_con1: node degree is higher than MAXCONNEC. Exit.");
      exit(1);
    }
    ep++;
  }
  for (i=1; i<=nnodes;i++)
    if (nodeticks[i]>ideg) ideg=nodeticks[i];
  free(nodeticks);
  if (ideg>MAXCONNEC)
  {
    perror(" max_con2: node degree is higher than MAXCONNEC. Exit.");
    exit(1);
  }
}



void read_data()
{
  FILE *fp;

  if((fp = fopen(node_in,"r")) != NULL)
  {
    rnode(fp);
  }
  else
  {
    (void)fprintf(stderr,"no node file\n");
    exit(1);
  }
  (void)fclose(fp);
```

```
  if((fp = fopen(elmt_in,"r")) != NULL)
    relmt(fp);
  else
  {
    (void)fprintf(stderr,"no elmt file\n");
    exit(1);
  }
  (void)fclose(fp);
}


void relmt(FILE *fp)
{
  int e0;
  int e1;
  double t;
  double a;
  double l;
  double s;
  double d;
  struct elem *el;
  int ne;
  int counter=1;
  char line[100];

  (void)fgets(line,100,fp);
  if(sscanf(line,"%*10c%5d",&nelmts) != 1) exit(1);
  ne = nelmts;

  if((elems = (struct elem *)malloc((unsigned)sizeof(struct elem)*ne)) ==
NULL)
  {
    perror("relmt-malloc"); exit(1);
  }

  (void)memset((char *)elems,0,sizeof(struct elem)*ne);

  for(el = elems; ne-- > 0; el++)
  {
    if(fgets(line,80,fp) == 0) exit(1);

(void)sscanf(line,"%*5c%5d%5d%12lf%12lf%12lf%12lf%12lf",&e0,&e1,&t,&a,&l,&s
,&d);
    el->n[0]   = e0;
    el->n[1]   = e1;
    el->transm = t;
    el->apert  = a;
    el->length = l;
    el->stor   = s;
    el->diff   = d;
//    printf("elmt(%d): n0: %3d; n1: %3d\n",counter++, e0, e1);
  }
  fprintf(stdout,"number of elements is %d.\n",nelmts);
}

void rnode(FILE *fp)
{
  double x;
  double y;
  double z;
  double bv;
  double cb;
  int s;
```

```
  int b;
  int bc;
  struct node *nd;
  int nn;
  char line[100];

  (void)fgets(line,80,fp);
  if(sscanf(line,"%*10c%5d",&nnodes) != 1) exit(1);
  n = nn = nnodes;
  fprintf(stdout,"%s %d\n","nnodes is ",nnodes);
  if (nnodes>MAXNODES)
  {
    perror("exceed maximum (hardcoded:MAXNODES) number of nodes. Exit.");
    exit(1);
  }
  if((nodes = (struct node *)malloc((unsigned)sizeof(struct node)*(nn+1)))
== NULL)
  {
    perror("rnode-malloc"); exit(1);
  }

  (void)memset((char *)nodes,0,sizeof(struct node)*(nn+1));

  for(nd = (nodes+1); nn-- > 0; nd++)
  {
    if(fgets(line,80,fp) == 0) exit(1);

(void)sscanf(line,"%*5c%3d%2d%2d%12lf%12lf%12lf%12lf%12lf",&s,&b,&bc,&x,&y,
&z,&bv,&cb);
    nd->isc = s;
    nd->ib  = b;
    nd->ob = b;
        /* bc is ignored */
    nd->x = x;
    nd->y = y;
    nd->z = z;
    nd->bvalu  = bv;
    nd->cbvalu = cb;

  }
}




void write_data()
{
  FILE *fp;

  if((fp = fopen(node_out,"w")) != NULL)
  {
    wnode(fp);
  }
  else
  {
    (void)fprintf(stderr,"cannot open node output file\n");
    exit(1);
  }
  (void)fclose(fp);

  if((fp = fopen(elmt_out,"w")) != NULL)
    welmt(fp);
```

```
  else
  {
    (void)fprintf(stderr,"cannot open elmt output file\n");
    exit(1);
  }
  (void)fclose(fp);
}


void welmt(FILE *fp)
{
  int ne, count;
  struct elem *ep;

  fprintf(fp,"           %4d\n",nelmts);
  for (count=0,ne=nelmts, ep=elems; ne-- >0; ep++)
  {
    fprintf(fp,"%5d%5d%5d% 12.5e% 12.5e% 12.5e% 12.5e%
12.5e\n",1+count++,renum[ep->n[0]],renum[ep->n[1]],
     ep->transm,ep->apert,ep->length, ep->stor, ep->diff);
  }
  return;
}


/*-------------------------------------------------*/

void wnode(FILE *fp)
{
  double x;
  double y;
  double z;
  double bv;
  double cb;
  int s;
  int b;
  int bc;
  struct node *nd;
  int i;
  for (i=1; i<= nnodes; i++)
    node_con[renum[i]]= i;

  fprintf(fp,"           %4d\n",nnodes);

  for(i = 1; i <= nnodes;i++)
  {
    nd = nodes+node_con[i];
    s = nd->isc;
    b = nd->ib;
    bc = 0;    /* bc is ignored */
    x = nd->x;
    y = nd->y;
    z = nd->z;
    bv = nd->bvalu;
    cb = nd->cbvalu;
    fprintf(fp,"%5d%3d%2d%2d% 12f% 12f% 12f% 12f%
12f\n",i,s,b,bc,x,y,z,bv,cb);
  }
}
```